

AD-A142 783

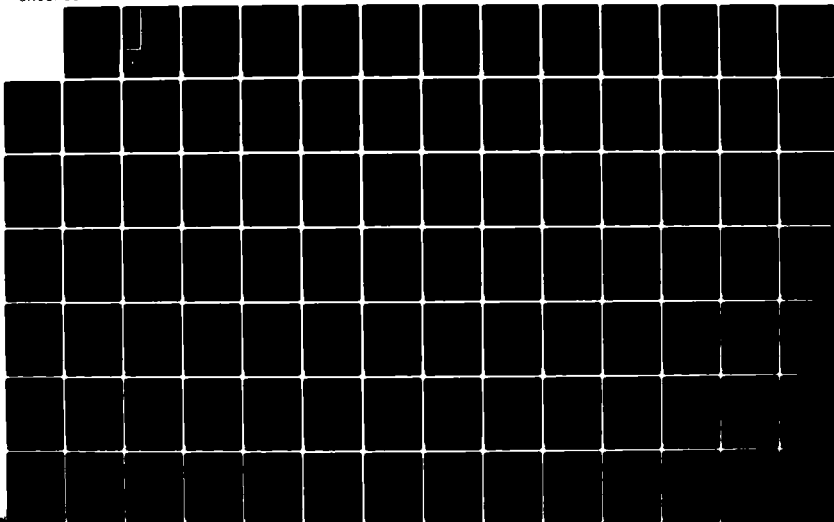
PROCEEDINGS PAPERS OF THE AFSC (AIR FORCE SYSTEMS
COMMAND) AVIONICS STAND..(U) AERONAUTICAL SYSTEMS DIV
WRIGHT-PATTERSON AFB OH DIRECTORATE O..
C A PORUBCANSKY NOV 82

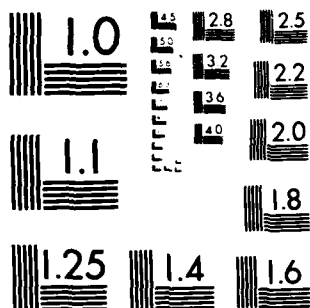
1/2

UNCLASSIFIED

F/G 9/2

NL





MICROCOPY RESOLUTION TEST CHART
NATIONAL BUREAU OF STANDARDS 1963-A

AD-A142 783

2nd AFSC STANDARDIZATION CONFERENCE

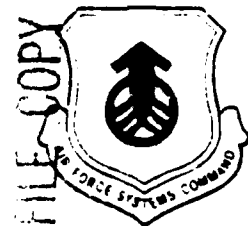
COMBINED PARTICIPATION BY:
DOD-ARMY-NAVY-AIR FORCE-NATO



30 NOVEMBER - 2 DECEMBER 1982
TUTORIALS: 29 NOVEMBER 1982

DAYTON CONVENTION CENTER
DAYTON, OHIO

SPONSORED BY



DTIC FILE COPY

TUTORIAL

MIL-STD-1815
ADA HIGH ORDER LANGUAGE

HOSTED BY



NOTICE

When Government drawings, specifications, or other data are used for any purpose other than in connection with a definitely related Government procurement operation, the United States Government thereby incurs no responsibility nor any obligation whatsoever, and the fact that the government may have formulated, furnished, or in any way supplied the said drawings, specifications, or other data, is not to be regarded by implication or otherwise as in any manner licensing the holder or any other person or corporation, or conveying any rights or permission to manufacture use, or sell any patented invention that may in any way be related thereto.

This report has been reviewed by the Office of Public Affairs (ASD/PA) and is releasable to the National Technical Information Service (NTIS). At NTIS, it will be available to the general public, including foreign nations.

This technical report has been reviewed and is approved for publication.

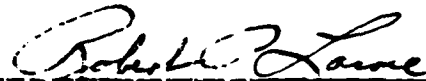


JEFFERY L. PESLER
Vice Chairman
And AFSC Standardization Conference



ERWIN C. GANGL
Chief, Avionics Systems Division
Directorate of Avionics Engineering

FOR THE COMMANDER



ROBERT P. LAVOIE, COL, USAF
Director of Avionics Engineering
Deputy for Engineering

"If your address has changed, if you wish to be removed from our mailing list, or if the addressee is no longer employed by your organization please notify ASD/ENAS, W-PAFB, OH 45433 to help us maintain a current mailing list".

Copies of this report should not be returned unless return is required by security considerations, contractual obligations, or notice on a specific document.

UNCLASSIFIED

SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

REPORT DOCUMENTATION PAGE		READ INSTRUCTIONS BEFORE COMPLETING FORM
1. REPORT NUMBER ASD(ENA)-TR-82-5031, VOLUME VIII	2. GOVT ACCESSION NO. AD A142753	3. RECIPIENT'S CATALOG NUMBER
4. TITLE (and Subtitle) Proceedings Papers of the Second AFSC Avionics Standardization Conference		5. TYPE OF REPORT & PERIOD COVERED Final Report 29 November - 2 December 1982
		6. PERFORMING ORG. REPORT NUMBER
7. AUTHOR(s) Editor: Cynthia A. Porubcansky		8. CONTRACT OR GRANT NUMBER(s)
9. PERFORMING ORGANIZATION NAME AND ADDRESS HQ ASD/ENAS Wright-Patterson AFB OH 45433		10. PROGRAM ELEMENT, PROJECT, TASK AREA & WORK UNIT NUMBERS
11. CONTROLLING OFFICE NAME AND ADDRESS HQ ASD/ENA Wright-Patterson AFB OH 45433		12. REPORT DATE November 1982
		13. NUMBER OF PAGES
14. MONITORING AGENCY NAME & ADDRESS (if different from Controlling Office) Same as Above		15. SECURITY CLASS. (of this report) Unclassified
		15a. DECLASSIFICATION, DOWNGRADING SCHEDULE
16. DISTRIBUTION STATEMENT (of this Report) Approved for public release; distribution unlimited.		
17. DISTRIBUTION STATEMENT (of the abstract entered in Block 20, if different from Report) N/A		
18. SUPPLEMENTARY NOTES N/A		
19. KEY WORDS (Continue on reverse side if necessary and identify by block number) Computer Instruction Set Architecture, Multiplexing, Compilers, Support Software, Data Bus, Rational Standardization, Digital Avionics, System Integration, Stores Interface, Standardization, MIL-STD-1553, MIL-STD-1589 (JOVIAL), MIL-STD-1750, MIL-STD-1760, MIL-STD-1815 (ADA), MIL-STD-1862 (NEBULA).		
20. ABSTRACT (Continue on reverse side if necessary and identify by block number) This is a collection of UNCLASSIFIED papers to be distributed to the attendees of the Second AFSC Avionics Standardization Conference at the Convention Center, Dayton, Ohio. The scope of the Conference includes the complete range of DoD approved embedded computer hardware/software and related interface standards as well as standard subsystems used within the Tri-Service community and NATO. The theme of the conference is "Rational Standardization". Lessons learned as well as the pros and cons of standardization are highlighted.		

DD FORM 1 JAN 73 1473

EDITION OF 1 NOV 65 IS OBSOLETE

UNCLASSIFIED

SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

This is Volume 8

Volume 1	Proceedings pp. 1-560
Volume 2	Proceedings pp. 561-1131
Volume 3	Governing Documents
Volume 4	MIL-STD-1553 Tutorial
Volume 5	MIL-STD-1589 Tutorial
Volume 6	MIL-STD-1679 Tutorial
Volume 7	MIL-STD-1750 Tutorial
Volume 8	MIL-STD-1815 Tutorial
Volume 9	Navy Case Study Tutorial

PROCEEDINGS OF THE

**2nd AFSC
STANDARDIZATION CONFERENCE**

30 NOVEMBER - 2 DECEMBER 1982

**DAYTON CONVENTION CENTER
DAYTON, OHIO**

Sponsored by:

Hosted by:

Air Force Systems Command

Aeronautical Systems Division

FOREWORD

THE UNITED STATES AIR FORCE HAS COMMITTED ITSELF TO "STANDARDIZATION." THE THEME OF THIS YEAR'S CONFERENCE IS "RATIONAL STANDARDIZATION," AND WE HAVE EXPANDED THE SCOPE TO INCLUDE US ARMY, US NAVY AND NATO PERSPECTIVES ON ONGOING DOD INITIATIVES IN THIS IMPORTANT AREA.

WHY DOES THE AIR FORCE SYSTEMS COMMAND SPONSOR THESE CONFERENCES? BECAUSE WE BELIEVE THAT THE COMMUNICATIONS GENERATED BY THESE GET-TOGETHERS IMPROVE THE ACCEPTANCE OF OUR NEW STANDARDS AND FOSTERS EARLIER, SUCCESSFUL IMPLEMENTATION IN NUMEROUS APPLICATIONS. WE WANT ALL PARTIES AFFECTED BY THESE STANDARDS TO KNOW JUST WHAT IS AVAILABLE TO SUPPORT THEM: THE HARDWARE; THE COMPLIANCE TESTING; THE TOOLS NECESSARY TO FACILITATE DESIGN, ETC. WE ALSO BELIEVE THAT FEEDBACK FROM PEOPLE WHO HAVE USED THEM IS ESSENTIAL TO OUR CONTINUED EFFORTS TO IMPROVE OUR STANDARDIZATION PROCESS. WE HOPE TO LEARN FROM OUR SUCCESSES AND OUR FAILURES; BUT FIRST, WE MUST KNOW WHAT THESE ARE AND WE COUNT ON YOU TO TELL US.

AS WE DID IN 1980, WE ARE FOCUSING OUR PRESENTATIONS ON GOVERNMENT AND INDUSTRY EXECUTIVES, MANAGERS, AND ENGINEERS AND OUR GOAL IS TO EDUCATE RATHER THAN PRESENT DETAILED TECHNICAL MATERIAL. WE ARE STRIVING TO PRESENT, IN A SINGLE FORUM, THE TOTAL AFSC STANDARDIZATION PICTURE FROM POLICY TO IMPLEMENTATION. WE HOPE THIS INSIGHT WILL ENABLE ALL OF YOU TO BETTER UNDERSTAND THE "WHY'S AND WHEREFORE'S" OF OUR CURRENT EMPHASIS ON THIS SUBJECT.

MANY THANKS TO A DEDICATED TEAM FROM THE DIRECTORATE OF AVIONICS ENGINEERING FOR ORGANIZING THIS CONFERENCE; FROM THE OUTSTANDING TECHNICAL PROGRAM TO THE UNGLAMOROUS DETAILS NEEDED TO MAKE YOUR VISIT TO DAYTON, OHIO A PLEASANT ONE. THANKS ALSO TO ALL THE MODERATORS, SPEAKERS AND EXHIBITORS WHO RESPONDED IN SUCH A TIMELY MANNER TO ALL OF OUR PLEAS FOR ASSISTANCE.

Robert P. Lavoie
ROBERT P. LAVOIE, COL, USAF
DIRECTOR OF AVIONICS ENGINEERING
DEPUTY FOR ENGINEERING

Accession For	
NTIS GRA&I	<input checked="" type="checkbox"/>
DTIC TAB	<input checked="" type="checkbox"/>
Unannounced	<input type="checkbox"/>
Justification	<input type="checkbox"/>
By	
Date	
Approved for release	
Distribution	
Dist	
A-1	

100-400000-2



DEPARTMENT OF THE AIR FORCE
HEADQUARTERS AIR FORCE SYSTEMS COMMAND
ANDREWS AIR FORCE BASE DC 20334

28 AUG 1982

REPLY TO
FORM OF

CV

SUBJECT

Second AFSC Standardization Conference

TO

ASD/OC

1. Since the highly successful standardization conference hosted by ASD in 1980, significant technological advancements have occurred. Integration of the standards into weapon systems has become a reality. As a result, we have many "lessons learned" and cost/benefit analyses that should be shared within the tri-service community. Also, this would be a good opportunity to update current and potential "users." Therefore, I endorse the organization of the Second AFSC Standardization Conference.

2. This conference should cover the current accepted standards, results of recent congressional actions, and standards planned for the future. We should provide the latest information on policy, system applications, and lessons learned. The agenda should accommodate both government and industry inputs that criticize as well as support our efforts. Experts from the tri-service arena should be invited to present papers on the various topics. Our AFSC project officer, Maj David Hammond, HQ AFSC/ALR, AUTOVON 858-5731, is prepared to assist.

ROBERT M. BOND, Lt Gen, USAF
Vice Commander

PRECEDING PAGE BLANK-NOT FILMED

MIL-STD-1815

ADA HIGH ORDER LANGUAGE

Instructor: Maj. Richard E. Bolz
U.S. Air Force Academy

ABSTRACT

This tutorial will discuss the development history, design and implementation of Mil Std 1815 (the Ada programming language). The syntax and semantics of the language will be covered in overview fashion with emphasis on data typing and the use of Ada as an object-oriented design language.

BIOGRAPHY

Major Richard E. Bolz has earned the BS and MS in Computer Science from Penn State University and has been a member of the Computer Science Department at the U.S. Air Force Academy since 1973. He is the co-developer of 'Software Engineering with Ada', A 4-day course for managers, analysts, designers and programmers.

PRECEDING PAGE BLANK-NOT FILMED

Software Engineering with Ada

As long as there were no machines, programming was no problem at all; when we had a few weak computers, programming became a mild problem and now that we have gigantic computers, programming has become an equally gigantic problem. In this sense the electronic industry has not solved a single problem, it has only created them - it has created the problem of using its product.

E. W. Dijkstra

Turing Award Lecture, 1972

SYMPTOMS OF THE SOFTWARE CRISIS

"...appear in the form of software that is non-responsive to user needs, unreliable, excessively expensive, untimely, inflexible, difficult to maintain, and not reusable."

David Fisher

Software Engineering with Ada

AS A REACTION TO THE SOFTWARE
CRISIS, THE U.S. DEPARTMENT OF
DEFENSE SPONSORED THE DEVELOP-
MENT OF THE ADA PROGRAMMING
LANGUAGE AND ITS ENVIRONMENT

PLANNING SECTION OF SSC COUNCIL

Software Engineering with Ada

THE ADA CULTURE

- * A PROGRAMMING LANGUAGE
- * A PROGRAMMING ENVIRONMENT
- * A WAY OF THINKING

Introduction

ADA

* REPRESENTS A "major advance in programming technology, bringing together the best ideas on the subject in a coherent way designed to meet the real needs of practical programmers." *I. C. Pyle*

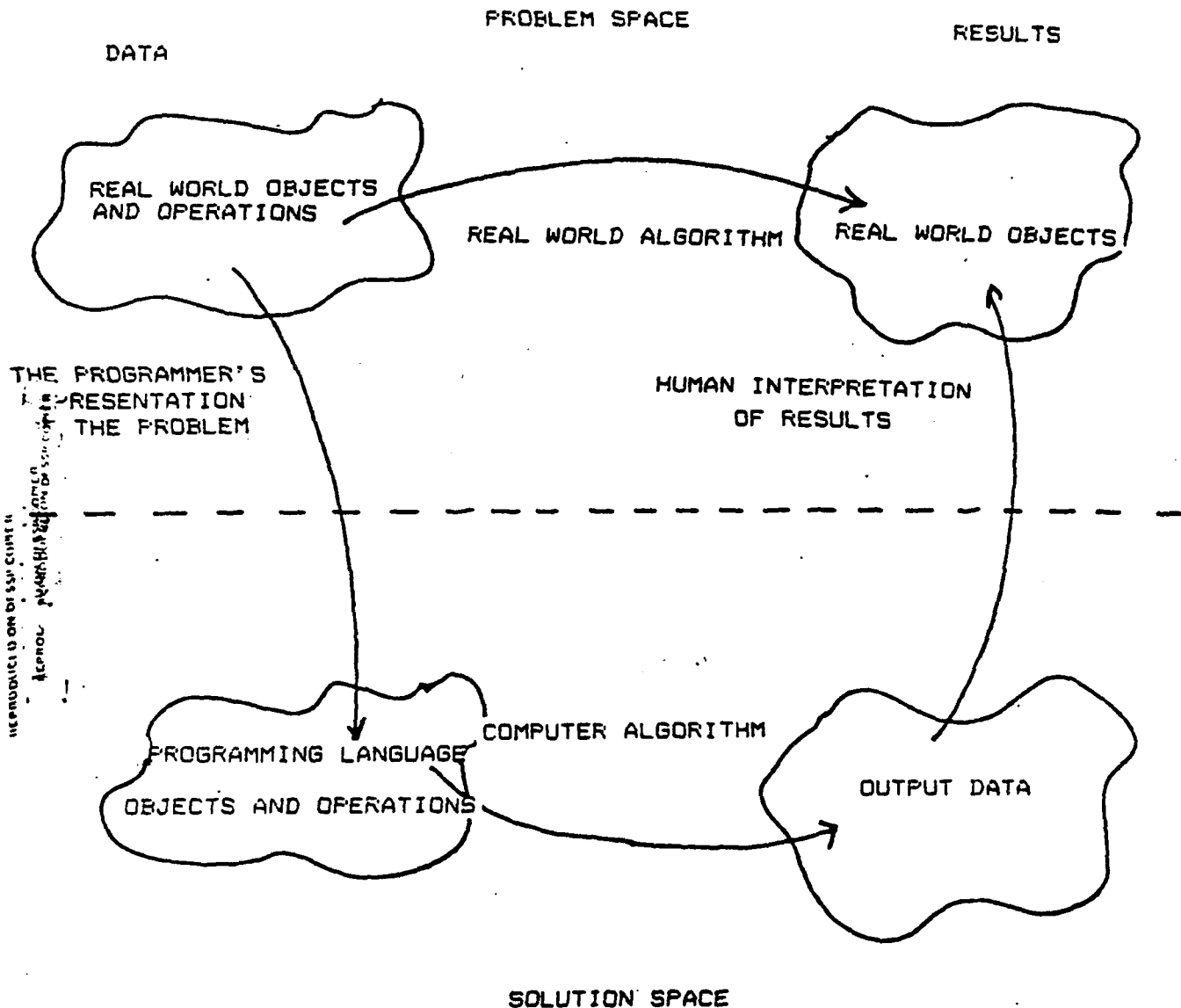
* IS A LANGUAGE THAT DIRECTLY EMBODIES AND ENFORCES MODERN SOFTWARE ENGINEERING PRINCIPLES

ABSTRACTION

- * EXTRACT ESSENTIAL DETAILS
- * OMIT INESSENTIAL DETAILS
- * EACH LEVEL OF DECOM-
POSITION REPRESENTS AN
ABSTRACTION
- * EACH LEVEL MUST BE COMPLETELY
UNDERSTOOD AS A UNIT
- * OUR VIEW OF THE WORLD FORMS
LADDERS OF ABSTRACTION

INFORMATION HIDING

- * MAKE DETAILS OF AN IMPLEMENTATION INACCESSIBLE
- * ENFORCE DEFINED INTERFACES
- * FOCUS ON THE ABSTRACTION OF AN OBJECT BY SUPPRESSING THE DETAILS
- * PREVENT HIGH LEVEL DECISIONS FROM BEING BASED ON LOW LEVEL CHARACTERISTICS



From the Programming Language Landscape by Henry Ledgard and Michael Marcotty
 (c) 1981 Science Research Associates, Inc. Reproduced by permission of
 the publisher

Figure 5-1: Model for a Typical Programming Task

ADA DESIGN GOALS

- * RECOGNITION OF THE IMPORTANCE OF PROGRAM RELIABILITY AND MAINTAINABILITY**
- * CONCERN FOR PROGRAMMING AS A HUMAN ACTIVITY**
- * EFFICIENCY**

**LANGUAGE REQUIREMENTS
(STEELMAN)**

- * STRUCTURED CONSTRUCTS**
- * STRONG TYPING**
- * RELATIVE AND ABSOLUTE
PRECISION SPECIFICATION**
- * INFORMATION HIDING AND
DATA ABSTRACTION**
- * CONCURRENT PROCESSING**
- * EXCEPTION HANDLING**
- * GENERIC DEFINITION**
- * MACHINE DEPENDENT
FACILITIES**

Software Engineering with Ada

ADA FROM THE TOP DOWN

* ADA SYSTEMS ARE COMPOSED OF

- SUBPROGRAMS
- PACKAGES
- TASKS

* ALL PROGRAM UNITS HAVE A TWO PART STRUCTURE

- SPECIFICATION (VISIBLE PART)
- BODY (HIDDEN PART)

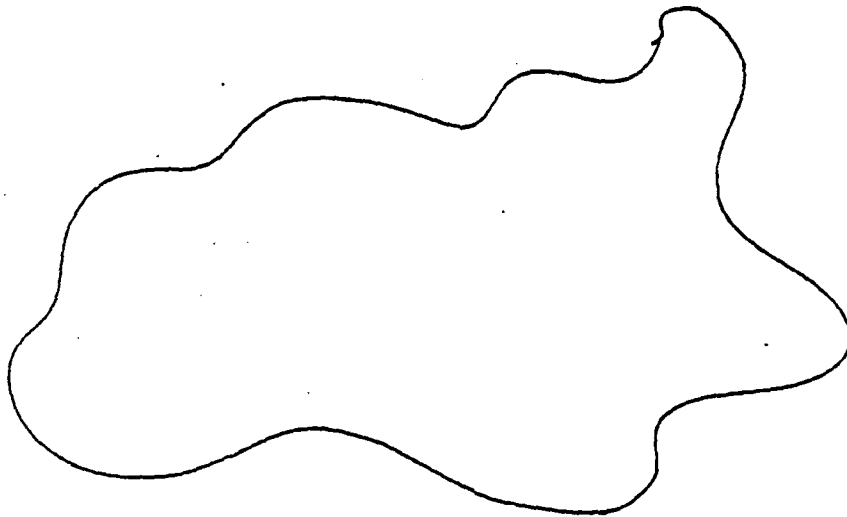


Figure 6-1: Symbol for an Undefined Entity

SUBPROGRAM SPECIFICATION



SUBPROGRAM BODY

Figure 6-2: Symbol for an Ada Subprogram

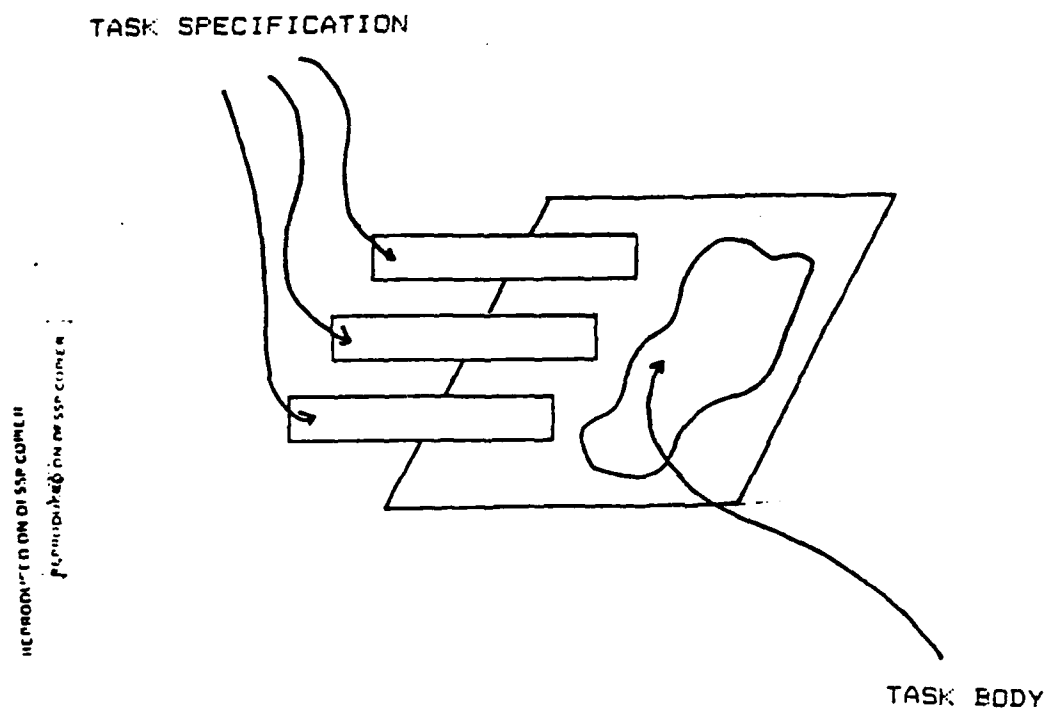


Figure 6-3: Symbol for an Ada Task

REPRODUCTION OF SSN CIPHER
REPRODUCTION OF SSN CIPHER

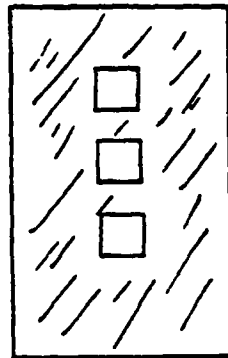


Figure 6-4: Package with Visible Parts

PACKAGE SPECIFICATION

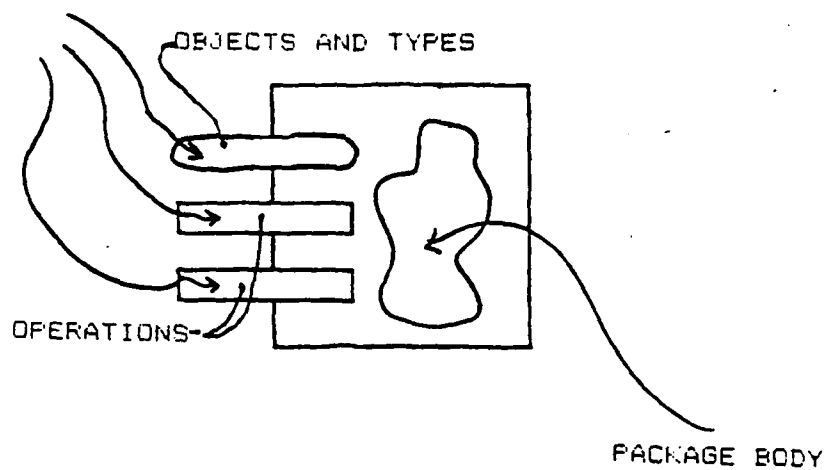


Figure 6-5: Symbol for an Ada Package

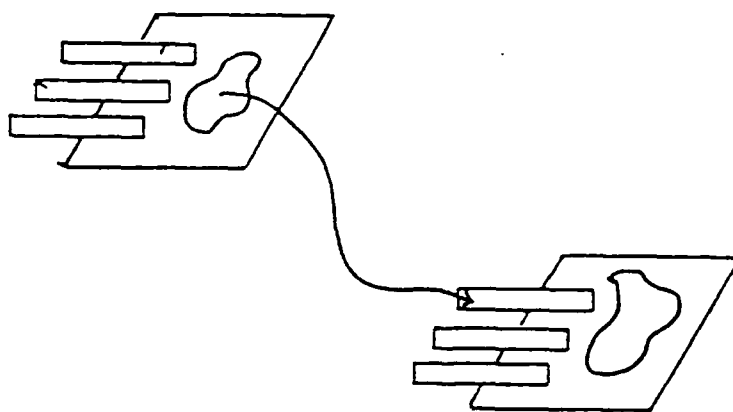


Figure 6-6: Communicating Ada Tasks

REPRODUCTION OF SSP CUMEN
PERMITTED BY NIA OF SSP CUMEN

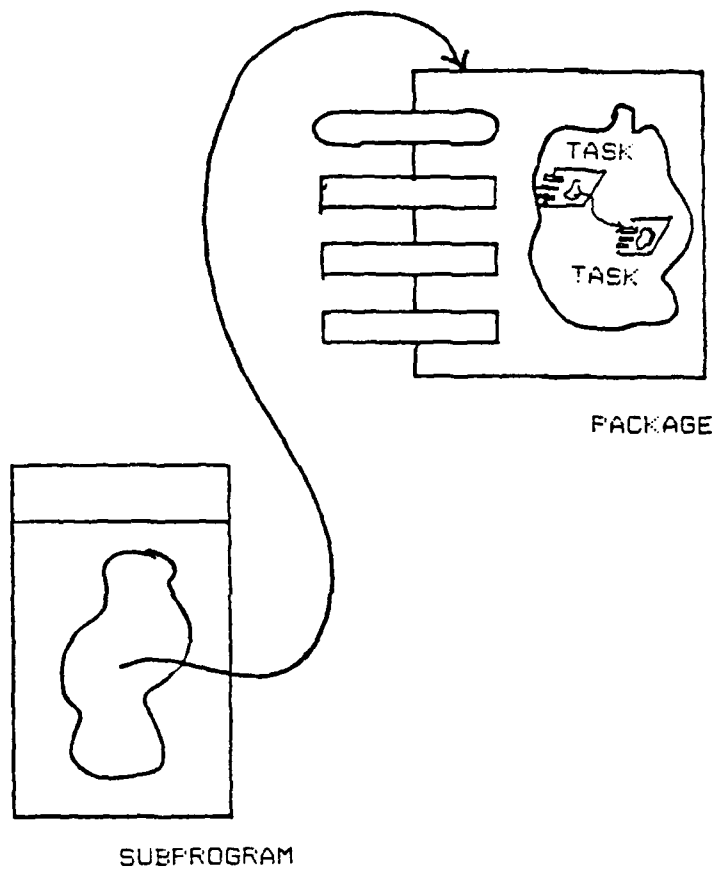


Figure 6-7: Nesting Ada Program Units

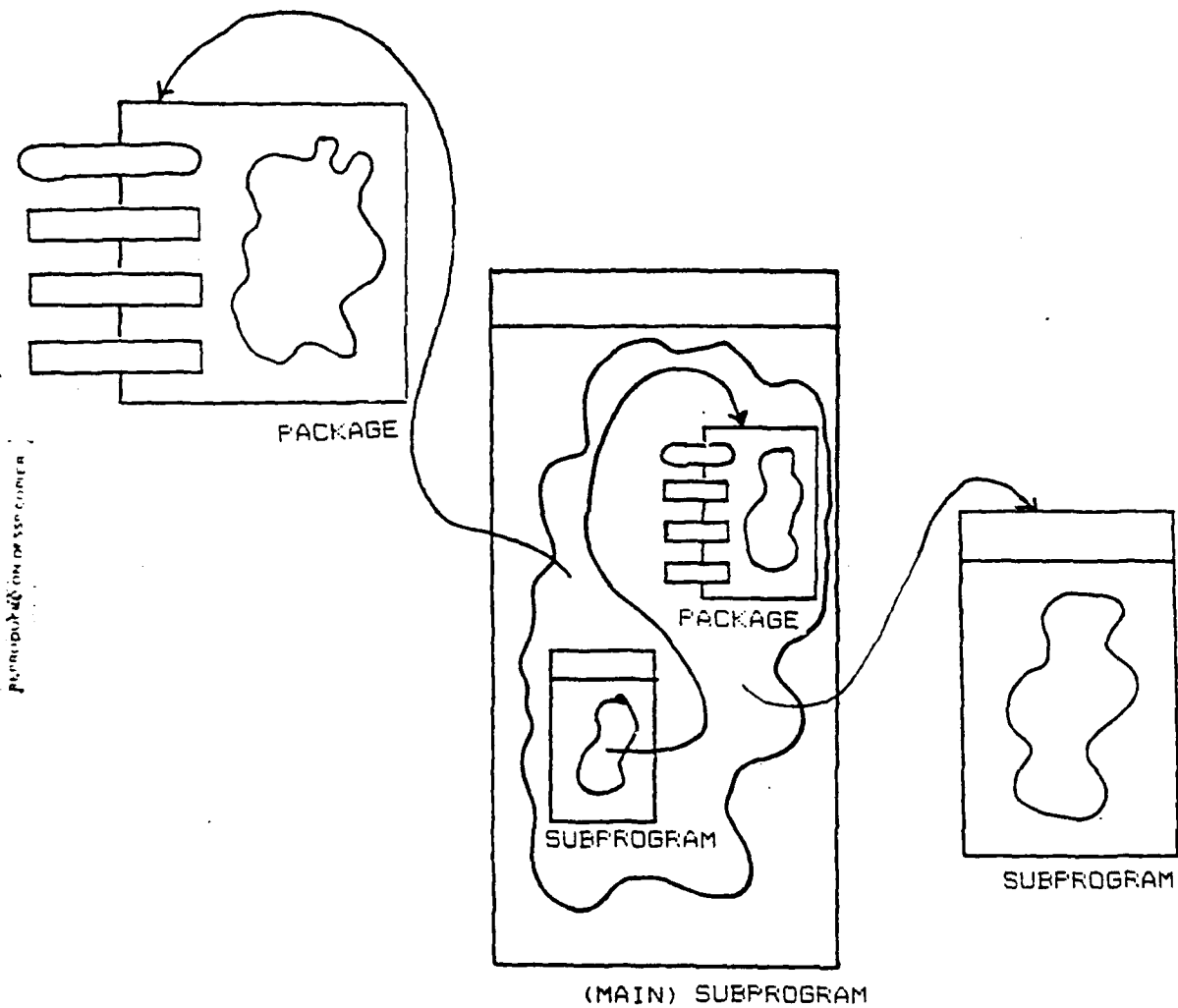


Figure 6-8: An Ada Program from the Top Down

ADA FROM THE BOTTOM UP

* ALL CONSTRUCTS ARE BUILT
FROM A STANDARD OR EXTENDED
CHARACTER SET (ASCII)

* LEXICAL UNITS INCLUDE

- IDENTIFIERS
- NUMERIC LITERALS
- CHARACTER LITERALS
- STRINGS
- DELIMITERS
- COMMENTS

Software Engineering with Ada

ADA RESERVED WORDS

abort	declare	generic	of	select
abs	delay	goto	or	separate
accept	delta		others	subtype
access	digits	if	out	
all	do	in		task
and		is	package	terminate
array			pragma	then
at	else		private	type
	elsif	limited	procedure	
	end	loop		
begin	entry		raise	use
body	exception		range	
	exit	mod	record	when
			rem	while
		new	renames	with
case	for	not	return	
constant	function	null	reverse	xor

TYPE DEFINITIONS

* A TYPE CHARACTERIZES

- A SET OF VALUES
- A SET OF OPERATIONS APPLICABLE TO THOSE VALUES

* ADA CLASSES OF TYPES INCLUDE

- SCALAR
 - INTEGER
 - REAL
 - ENUMERATION
- COMPOSITE
 - ARRAY
 - RECORD
- ACCESS
- PRIVATE
- SUBTYPE AND DERIVED TYPE

DECLARATIONS

* CREATE OBJECTS OF A GIVEN
TYPE

* ADA DECLARATIONS PERMIT

- VARIABLES
- CONSTANTS
- DYNAMIC CREATION

NAMES

- * DENOTE DECLARED ENTITIES
- * MAY BE OVERLOADED
- * ARE STRONGLY TYPED

OPERATORS AND EXPRESSIONS

* PREDEFINED OPERATORS INCLUDE

-- EXPONENTIATING	**		
-- MULTIPLYING	*	/	
	mod	rem	
-- UNARY	+	-	
	not	abs	
-- ADDING	+	-	&
-- RELATIONAL	=	/=	<
	<=	>	>=
-- LOGICAL	and	or	xor
	and then	or else	
-- MEMBERSHIP	in	not in	

* OPERATOR SYMBOLS MAY BE OVERLOADED

STATEMENTS

* PROVIDE CONTROL AND ACTION

* ADA STATEMENTS INCLUDE

-- SEQUENTIAL

- ASSIGNMENT
- NULL
- SUBPROGRAM CALL
- RETURN
- BLOCK

-- CONDITIONAL

- IF
- CASE

-- ITERATIVE

- LOOP
- EXIT

-- OTHER STATEMENTS

- ENTRY CALL - RAISE
- ACCEPT - CODE
- ABORT - GOTO
- DELAY
- SELECT

PERMISSION TO REPRODUCE THIS

SUBPROGRAMS

* ARE THE BASIC EXECUTABLE UNIT

* ADA SUBPROGRAMS INCLUDE

-- PROCEDURES

-- FUNCTIONS

PACKAGES

- * PERMIT THE COLLECTION OF
GROUPS OF LOGICALLY RELATED
ENTITIES
- * DIRECTLY SUPPORT INFORMATION
HIDING AND ABSTRACTION
- * PERMIT AN INDUSTRY OF
SOFTWARE MODULES

TASKS

- * PERMIT COMMUNICATING
SEQUENTIAL PROCESSES
- * USE THE CONCEPT OF A
RENDEZVOUS
- * SPECIAL STATEMENTS ARE
PROVIDED FOR TASK CONTROL

EXCEPTION HANDLING

- * PERMITS ERRORS TO BE CAPTURED FOR GRACEFUL DEGRADATION
- * IS BLOCK STRUCTURED
- * EXCEPTIONS MAY BE PRE-DEFINED OR USER DEFINED

GENERIC PROGRAM UNITS

- * DEFINE HIGH LEVEL TEMPLATES
- * PERMIT PARAMETERIZATION
OF SUBPROGRAMS AND PACKAGES
- * ENCOURAGE GENERAL PURPOSE
SOFTWARE LIBRARIES

REPRESENTATION SPECIFICATIONS

* PERMIT MAPPING THE LANGUAGE
TO THE UNDERLYING MACHINE

* INCLUDE SPECIFICATION OF

- LENGTH
- ENUMERATION TYPE REPRESENTATION
- RECORD TYPE REPRESENTATION
- ADDRESS SPECIFICATION

* PERMIT ACCESS TO

- INTERRUPTS
- IMPLEMENTATION DEPENDENT FEATURES

INPUT/OUTPUT

* ACHIEVED THROUGH THE PACKAGE FACILITY

* PREDEFINED I/O PACKAGES

-- HIGH-LEVEL IO

- SEQUENTIAL_IO

- DIRECT_IO

- TEXT_IO

-- LOW_LEVEL IO

**SUMMARY OF LANGUAGE
CHARACTERISTICS**

PLANNING IN THE DESIGN

- * GENERAL PURPOSE
- * EMPHASIS ON RELIABILITY
AND MAINTAINABILITY
- * DIRECTED TOWARD EFFICIENT
USE FOR LARGE, FREQUENTLY
MODIFIED SYSTEMS
- * INCORPORATES THE BEST OF
EXISTING SOFTWARE TECHNOLOGY
- * ENCOURAGES AND ENFORCES
SOFTWARE ENGINEERING
PRINCIPLES

In the development of our understanding of complex phenomena, the most powerful tool available to the human intellect is abstraction. Abstraction arises from a recognition of similarities between certain objects, situations or processes in the real world, and the decision to concentrate on these similarities, and to ignore the differences.

C.A.R. Hoare

Notes on Data Structuring

DEFINE THE PROBLEM

- * GIVEN A BINARY TREE,
COUNT ITS LEAVES

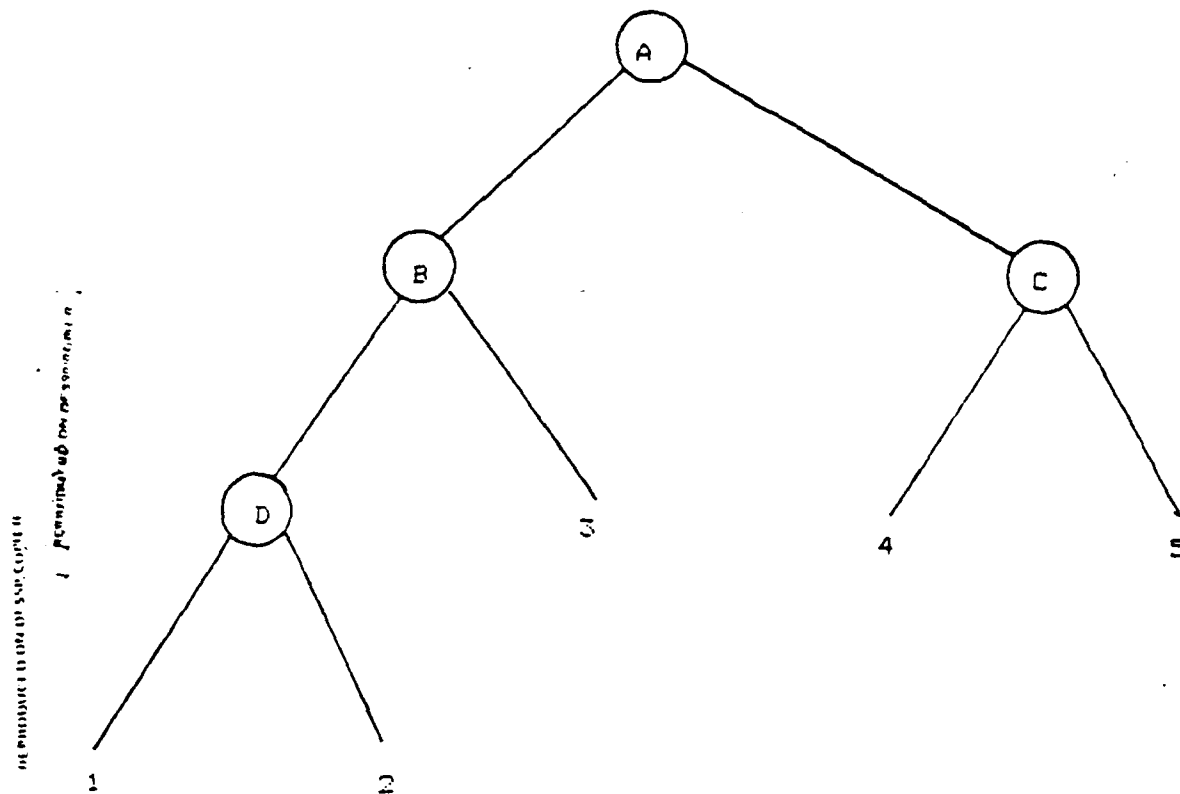


Figure 7-1: A Binary Tree

COUNTING LEAVES

* IF THE TREE IS A LEAF

NUMBER_OF_LEAVES (TREE) = 1

* IF THE TREE CONSISTS
OF TWO SUBTREES

NUMBER_OF_LEAVES (TREE) =
NUMBER_OF_LEAVES (RIGHT_SUBTREE) +
NUMBER_OF_LEAVES (LEFT_SUBTREE)

ASSUMPTIONS ABOUT THE SOLUTION

- * THE IMPLEMENTATION
LANGUAGE CONTAINS BASIC
CONTROL STRUCTURES

- SEQUENTIAL
 - CONDITIONAL
 - ITERATIVE

- * THERE ARE NO PREDEFINED
OBJECTS OR OPERATIONS

- * THE IMPLEMENTATION
LANGUAGE HAS FACILITIES
FOR CREATING OBJECTS
AND OPERATIONS

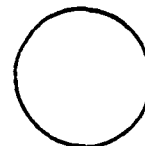
DEVELOP AN INFORMAL STRATEGY

KEEP A PILE OF THE PARTS OF THE TREE THAT HAVE NOT YET BEEN COUNTED. INITIALLY, GET A TREE AND PUT IT ON THE EMPTY PILE; THE COUNT OF THE LEAVES IS INITIALLY SET TO ZERO. AS LONG AS THE PILE IS NOT EMPTY, REPEATEDLY TAKE A TREE OFF THE PILE AND EXAMINE IT. IF THE TREE CONSISTS OF A SINGLE LEAF, THEN INCREMENT THE LEAF COUNTER AND THROW AWAY THAT TREE. IF THE TREE IS NOT A SINGLE LEAF BUT INSTEAD CONSISTS OF TWO SUBTREES, SPLIT THE TREE INTO ITS LEFT AND RIGHT SUBTREES AND PUT THEM BACK ON THE PILE. ONCE THE PILE IS EMPTY, DISPLAY THE COUNT OF THE LEAVES.

10. Take a tree off the pile
and examine it

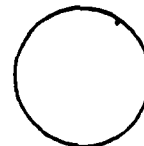
2

2



11. Since it is a leaf, count it
and throw away the tree

3



12. Since the pile is empty,
we can display the count

3

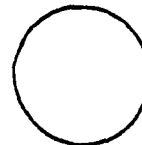
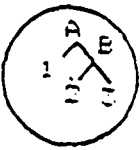
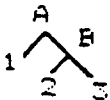
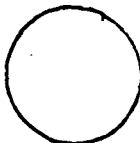

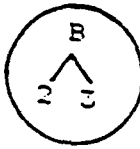
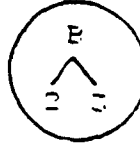
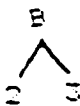
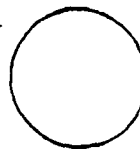
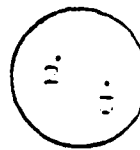

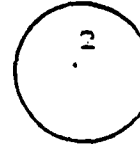


Figure 7-2: Example of Counting the Leaves

	LEAF_COUNT	TREE	FILE
1. Initially:	0		
2. Take a tree off the pile and examine it	0		
3. Since it is a tree, split it and return the subtrees	0		
4. Take a tree off the pile and examine it	0	1	
5. Since it is a leaf, count it and throw away the tree	1		
6. Take a tree off the pile and examine it	1		
7. Since it is a tree, split it and return the subtrees	1		
8. Take a tree off the pile and examine it	1		
9. Since it is a leaf, count it and throw away the tree	2		

FORMALIZE THE STRATEGY
IDENTIFY OBJECTS AND THEIR
ATTRIBUTES

KEEP A PILE OF THE PARTS OF THE TREE THAT HAVE NOT YET BEEN COUNTED. INITIALLY, GET A TREE AND PUT IT ON THE EMPTY PILE ; THE COUNT OF THE LEAVES IS INITIALLY SET TO ZERO. AS LONG AS THE PILE IS NOT EMPTY, REPEATEDLY TAKE A TREE OFF THE PILE AND EXAMINE IT. IF THE TREE CONSISTS OF A SINGLE LEAF , THEN INCREMENT THE LEAF COUNTER AND THROW AWAY THAT TREE. IF THE TREE IS NOT A SINGLE LEAF BUT INSTEAD CONSISTS OF TWO SUBTREES, SPLIT THE TREE INTO ITS LEFT AND RIGHT SUBTREES AND PUT THEM BACK ON THE PILE. ONCE THE PILE IS EMPTY, DISPLAY THE COUNT OF THE LEAVES.

Software Engineering with Ada

THE OBJECTS OF INTEREST ARE

- * LEAF_COUNT
- * PILE
- * LEFT_SUBTREE
RIGHT_SUBTREE
TREE

The First Design Problem

FORMALIZE THE STRATEGY
IDENTIFY OPERATIONS ON
THE OBJECTS

KEEP A PILE OF THE PARTS OF THE TREE THAT HAVE NOT YET BEEN COUNTED. INITIALLY, GET A TREE AND PUT IT ON THE EMPTY PILE; THE COUNT OF THE LEAVES IS INITIALLY SET TO ZERO. AS LONG AS THE PILE IS NOT EMPTY, REPEATEDLY TAKE A TREE OFF THE PILE AND EXAMINE IT. IF THE TREE CONSISTS OF A SINGLE LEAF, THEN INCREMENT THE LEAF COUNTER AND THROW AWAY THAT TREE. IF THE TREE IS NOT A SINGLE LEAF BUT INSTEAD CONSISTS OF TWO SUBTREES, SPLIT THE TREE INTO ITS LEFT AND RIGHT SUBTREES AND PUT THEM BACK ON THE PILE. ONCE THE PILE IS EMPTY, DISPLAY THE COUNT OF THE LEAVES.

THE OPERATIONS OF INTEREST ARE

* LEAF_COUNT

-- DISPLAY
-- INCREMENT
-- ZERO

* PILE

-- IS_NOT_EMPTY
-- PUT
-- PUT_INITIAL
-- TAKE

* LEFT_SUBTREE
RIGHT_SUBTREE
TREE

-- GET_INITIAL
-- IS_SINGLE_LEAF
-- THROW_AWAY

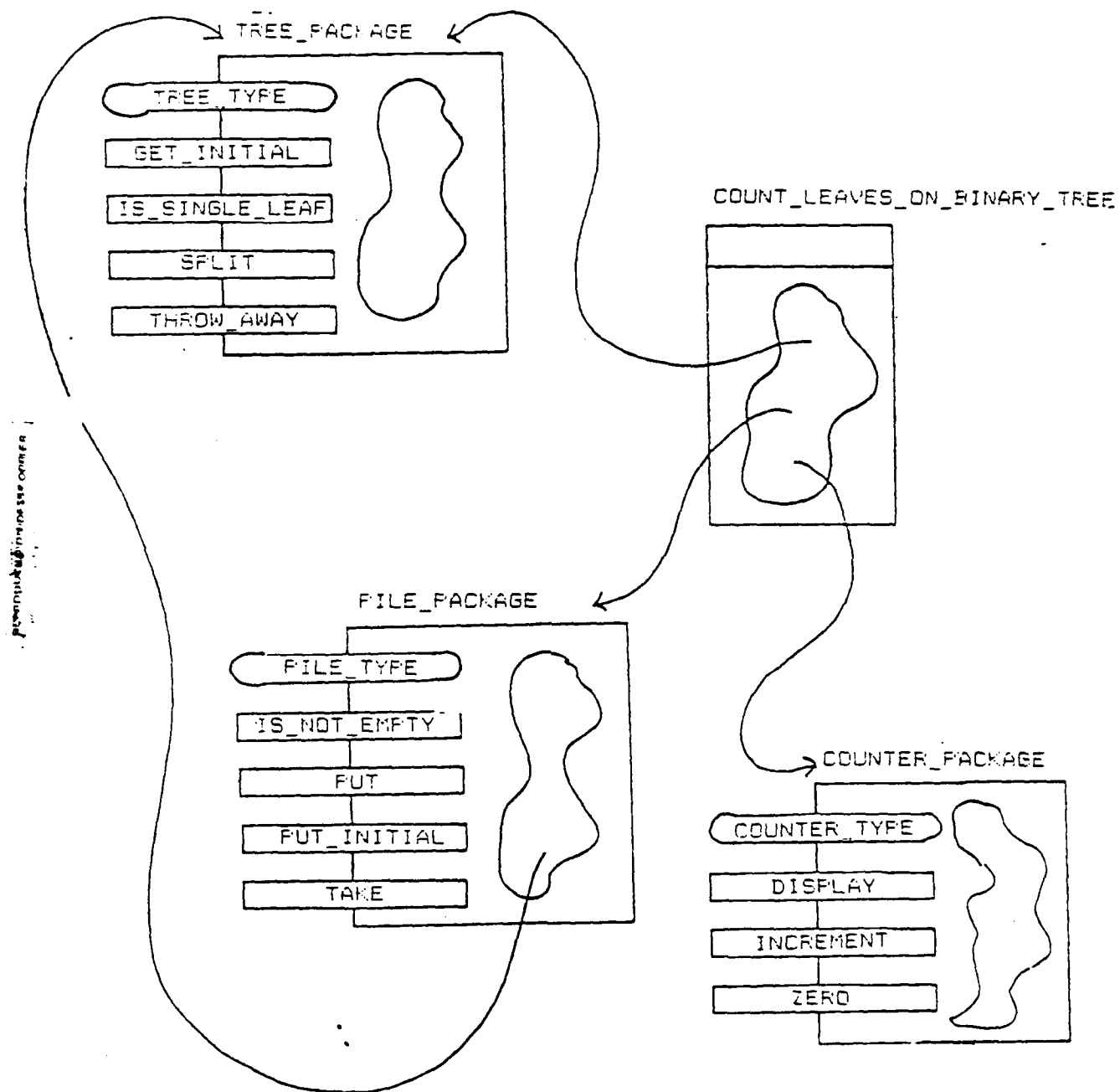


Figure 7-3: Design of `COUNT_LEAVES_ON_BINARY_TREE`

FORMALIZE THE STRATEGY
ESTABLISH THE INTERFACES

```
package COUNTER_PACKAGE is
  type COUNTER_TYPE is limited private;
  procedure DISPLAY (COUNTER : in    COUNTER_TYPE);
  procedure INCREMENT(COUNTER : in out COUNTER_TYPE);
  procedure ZERO      (COUNTER : out  COUNTER_TYPE);
private
  ...
end COUNTER_PACKAGE;
```

**FORMALIZE THE STRATEGY
ESTABLISH THE INTERFACES**

with TREE_PACKAGE;

package PILE_PACKAGE is

 type PILE_TYPE is limited private;

 function IS_NOT_EMPTY(PILE : in PILE_TYPE) return BOOLEAN;

 procedure PUT (TREE : in out TREE_PACKAGE.TREE_TYPE;
 ON : in out PILE_TYPE);

 procedure PUT_INITIAL (TREE : in out TREE_PACKAGE.TREE_TYPE;
 ON : in out PILE_TYPE);

 procedure TAKE (TREE : out TREE_PACKAGE.TREE_TYPE;
 OFF : in out PILE_TYPE);

private

 ...

end PILE_PACKAGE;

FORMALIZE THE STRATEGY
ESTABLISH THE INTERFACES

```
package TREE_PACKAGE is
  type TREE_TYPE is limited private;
  procedure GET_INITIAL  (TREE      : out   TREE_TYPE);
  function  IS_SINGLE_LEAF(TREE      : in    TREE_TYPE)
                                return BOOLEAN;
  procedure SPLIT        (TREE      : in out TREE_TYPE;
                           LEFT_INT0 : out   TREE_TYPE;
                           RIGHT_INT0: out   TREE_TYPE);
  procedure THROW_AWAY   (TREE      : in out TREE_TYPE);
private
  ..
end TREE_PACKAGE;
```

**FORMALIZE THE STRATEGY
IMPLEMENT THE OPERATIONS**

with COUNTER_PACKAGE, PILE_PACKAGE, TREE_PACKAGE;

use COUNTER_PACKAGE, PILE_PACKAGE, TREE_PACKAGE;

procedure COUNT_LEAVES_ON_BINARY_TREE is

LEAF_COUNT : COUNTER_TYPE;

LEFT_SUBTREE : TREE_TYPE;

PILE : PILE_TYPE;

RIGHT_SUBTREE : TREE_TYPE;

TREE : TREE_TYPE;

Software Engineering with Ada

```
begin
  GET_INITIAL(TREE);
  PUT_INITIAL(TREE, ON => PILE);
  ZERO(LEAF_COUNT);
  while IS_NOT_EMPTY(PILE)
    loop
      TAKE(TREE, OFF => PILE);
      if IS_SINGLE_LEAF(TREE) then
        INCREMENT(LEAF_COUNT);
        THROW_AWAY(TREE);
      else
        SPLIT(TREE,
              LEFT_INT0 => LEFT_SUBTREE,
              RIGHT_INT0 => RIGHT_SUBTREE);
        PUT(LEFT_SUBTREE, ON => PILE);
        PUT(RIGHT_SUBTREE, ON => PILE);
      end if;
    end loop;
  DISPLAY(LEAF_COUNT);
end COUNT_LEAVES_ON_BINARY_TREE;
```

The First Design Problem

DATA TYPES ADDRESS

* MAINTAINABILITY

- THE NEED TO DESCRIBE OBJECTS WITH A FACTORIZATION
OF PROPERTIES

* READABILITY

- THE NEED TO SAY SOMETHING ABOUT THE PROPERTIES
OF OBJECTS

* RELIABILITY

- THE NEED TO GUARANTEE THAT PROPERTIES OF
OBJECTS ARE NOT VIOLATED

* REDUCTION OF COMPLEXITY

- THE NEED TO HIDE IMPLEMENTATION DETAILS

Software Engineering with Ada

A TYPE CHARACTERIZES

- * A SET OF VALUES
- * A SET OF OPERATIONS
APPLICABLE TO OBJECTS
OF THE NAMED TYPE

permitted in the course

ADA CLASSES OF TYPES INCLUDE

* SCALAR

— THE VALUES HAVE NO COMPONENTS

* COMPOSITE

— THE VALUES CONSIST OF COMPONENT OBJECTS

* ACCESS

— THE VALUES PROVIDE ACCESS TO OTHER OBJECTS

* PRIVATE

— THE VALUES ARE NOT KNOWN TO A USER

* SUBTYPE AND DERIVED TYPE

* TASK TYPE

INTEGER TYPES

* INTRODUCE A SET OF
CONSECUTIVE EXACT INTEGERS

* USER-DEFINED TYPES

type LINE_COUNT is range 0 .. 66;
type INDEX is range 55 .. 77;
type FATHOM is range -5000 .. 0;
type TOTAL_ELEMENTS is range 1 .. (ROWS*COLUMNS);

SUMMARY OF INTEGER DATA TYPES

* SET OF VALUES

-- A SET OF CONSECUTIVE INTEGERS

* STRUCTURE

-- range L .. U

WHERE L AND U ARE STATIC EXPRESSIONS

REPRESENTING LOWER AND UPPER BOUNDS

* SET OF OPERATIONS

-- ADDING	+	-		
-- ASSIGNMENT	:=			
-- EXPONENTIATING	**			
-- EXPLICIT CONVERSION				
-- MEMBERSHIP	in	not in		
-- MULTIPLYING	*	/	mod	rem
-- QUALIFICATION				
-- RELATIONAL	=	/=	<	<=
	>	>=		
-- UNARY	+	-	abs	

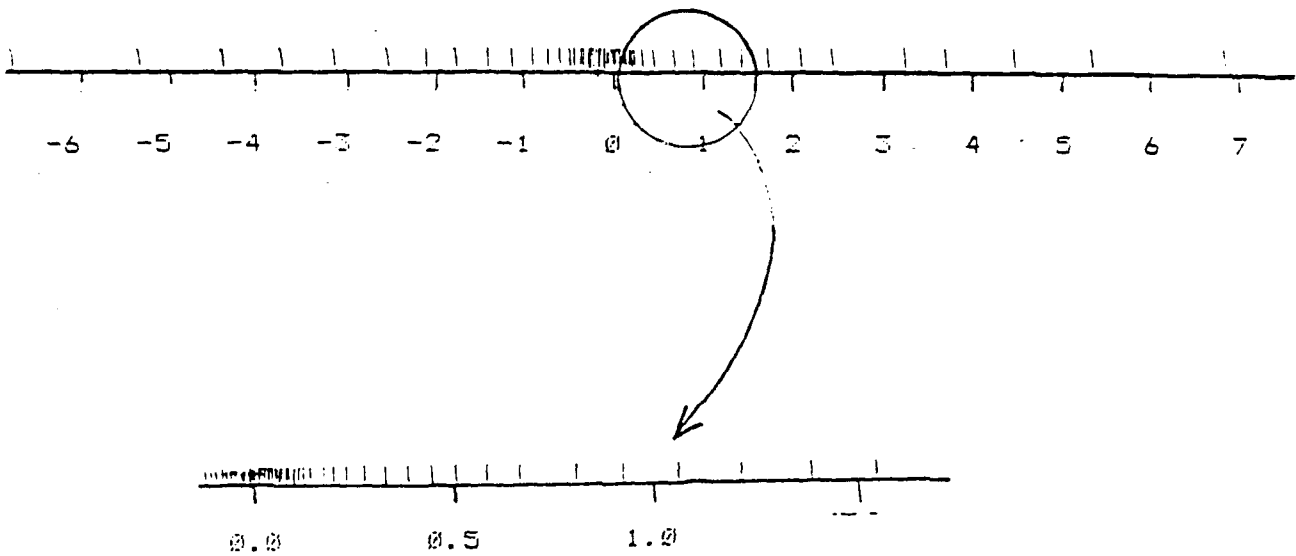


Figure 8-3: Floating Point Model Numbers

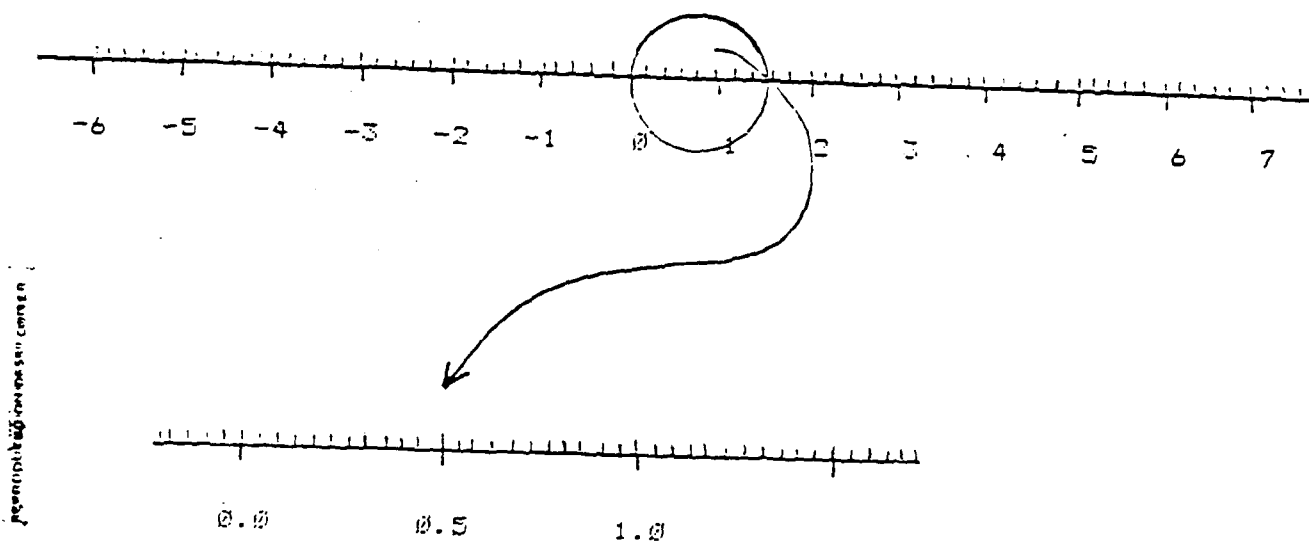


Figure E-4: Fixed Point Model Numbers

SUMMARY OF REAL DATA TYPES

* SET OF VALUES

-- APPROXIMATIONS TO THE REAL NUMBERS

* STRUCTURE

-- digits N range L .. U

SPECIFIES RELATIVE ACCURACY

WHERE N IS A STATIC INTEGER REPRESENTING

THE NUMBER OF SIGNIFICANT DIGITS AND

WHERE L AND U ARE STATIC EXPRESSIONS

REPRESENTING LOWER AND UPPER BOUNDS

-- delta N range L .. U

SPECIFIES ABSOLUTE ACCURACY

WHERE N IS A STATIC REAL VALUE

REPRESENTING THE DELTA AND

WHERE L AND U ARE STATIC EXPRESSIONS

REPRESENTING LOWER AND UPPER BOUNDS

*** SET OF OPERATIONS**

-- ADDING + -
-- ASSIGNMENT :=
-- EXPONENTIATING **
-- EXPLICIT CONVERSION
-- MEMBERSHIP in not in
-- MULTIPLYING * / mod rem
-- QUALIFICATION
-- RELATIONAL = /= < <=
 > >=
-- UNARY + - abs

*** ATTRIBUTES**

-- FIXED POINT ATTRIBUTES
ADDRESS SIZE
BASE MACHINE_OVERFLOW
FIRST SAFE_SMALL
LAST SAFE_LARGE

-- FLOATING POINT ATTRIBUTES

ADDRESS	MACHINE_MANTISSA
BASE	MACHINE_OVERFLOWS
DIGITS	MACHINE_RADIX
EMAX	MACHINE_ROUNDS
EPSILON	MANTISSA
FIRST	SAFE_EMAX
LARGE	SAFE_LARGE
LAST	SAFE_SMALL
MACHINE_EMAX	SIZE
MACHINE_EMIN	SMALL

* PREDEFINED TYPES

DURATION
FLOAT
LONG_FLOAT
SHORT_FLOAT

ENUMERATION TYPES

* INTRODUCE AN ORDERED SET
OF DISTINCT VALUES

* USER-DEFINED TYPES

type CARD_SUIT is (CLUBS, DIAMONDS,
HEARTS, SPADES);

type GEAR_POSITION is (DOWN, UP);

type MOTOR_STATE is (OFF, FORWARD, REVERSE);

type HEX_DIGIT is ('A', 'B', 'C', 'D', 'E', 'F');

SUMMARY OF ENUMERATION TYPES

* SET OF VALUES

-- ORDERED SET OF DISTINCT VALUES

* STRUCTURE

-- (E_0, E_1, ... E_n)

WHERE E_i IS AN ORDERED ENUMERATION

LITERAL

* SET OF OPERATIONS

-- ASSIGNMENT :=

-- MEMBERSHIP in not in

-- QUALIFICATION

-- RELATIONAL = /= < <=
 > >=

*** ATTRIBUTES**

ADDRESS	PRED
BASE	SIZE
FIRST	SUCC
IMAGE	VAL
LAST	VALUE
POS	WIDTH

*** PREDEFINED TYPES**

BOOLEAN
CHARACTER

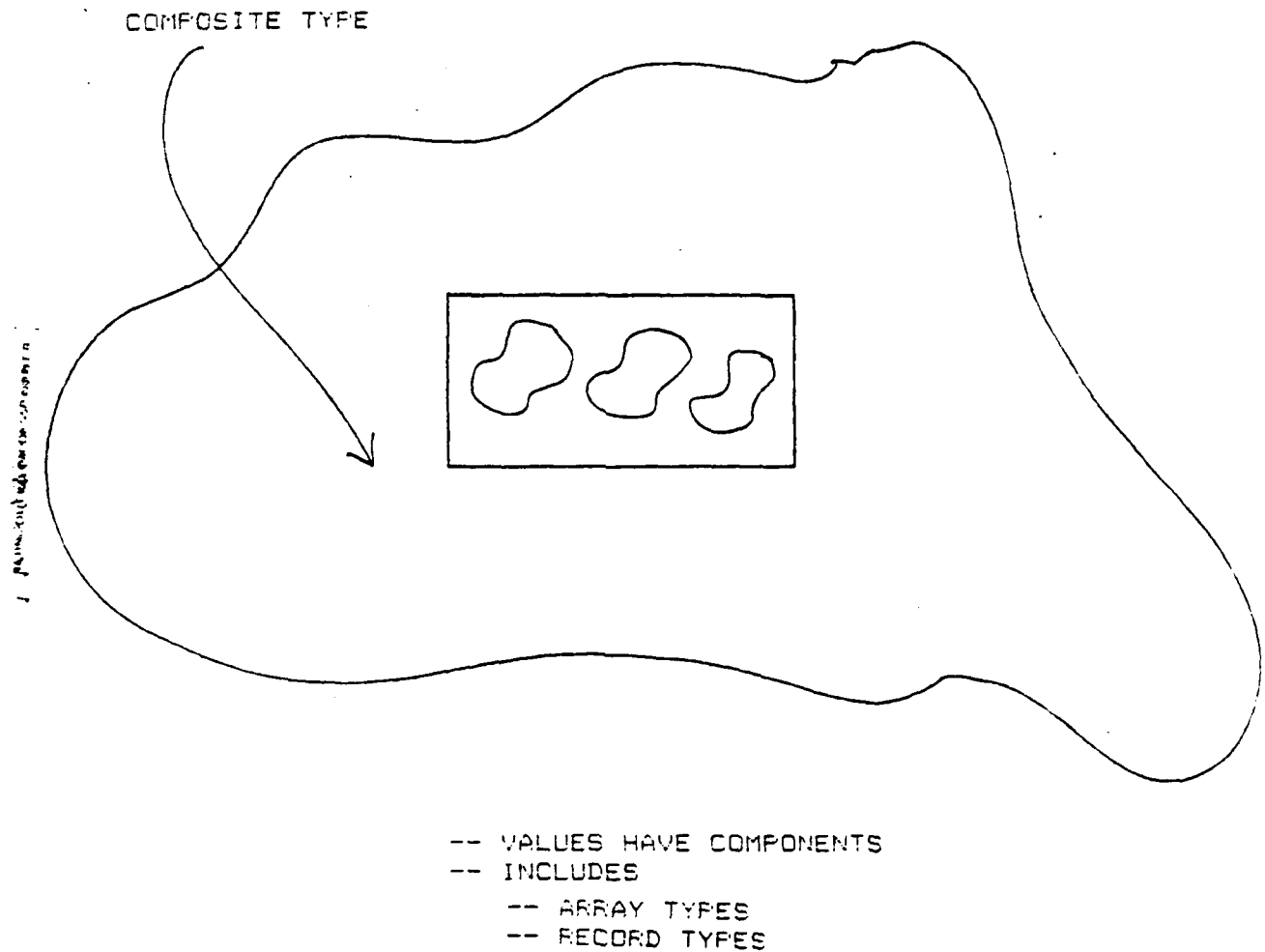


Figure 8-5: Composite Data Types

ARRAY TYPES

* INTRODUCE AN INDEXED COLLECTION OF SIMILAR TYPES

* CONSTRAINED TYPES

type GAME_BOARD is array (1 .. 8, 1 .. 8) of CHESS_PIECES;

type LIST is array (INTEGER range -100 .. 10) of FLOAT;

type VECTOR is array (INTEGER range 1 .. MAXIMUM_INDEX)
of FLOAT;

type LONG_ARRAY is array (EXTENDED_INDEX) of FLOAT;

type SHORT_ARRAY is array (EXTENDED_INDEX range 10 .. 49) of FLOAT;

type RECORD_OF_WORK is array (DAY range MONDAY .. FRIDAY) of HOURS;

type OVERTIME is array (DAY range SATURDAY .. SUNDAY) of HOURS;

type FULL_WEEK is array (DAY) of HOURS;

* UNCONSTRAINED TYPES

type BIT_VECTOR is array (INDEX range <>) of BOOLEAN;

type MATRIX is array (INDEX range <>, INDEX range <>) of FLOAT;

SUMMARY OF ARRAY TYPES

* SET OF VALUES

-- INDEXED COLLECTION OF SIMILAR TYPES

* STRUCTURE

-- array(INDEX...) of COMPONENT

AN UNCONSTRAINED ARRAY TYPE

WHERE INDEX... IS A LIST OF

UNCONSTRAINED DISCRETE TYPES

-- array INDEX_CONSTRAINT of COMPONENT

A CONSTRAINED ARRAY TYPE

WHERE INDEX_CONSTRAINT IS A LIST

OF DISCRETE TYPES

*** SET OF OPERATIONS**

-- ADDING &
-- AGGREGATES
-- ASSIGNMENT :=
-- EXPLICIT CONVERSION
-- INDEXING
-- LOGICAL and or xor
-- MEMBERSHIP in not in
-- QUALIFICATION
-- RELATIONAL = /= <
 <= > >=
-- UNARY not

*** ATTRIBUTES**

ADDRESS	LAST(J)
BASE	LENGTH
FIRST	LENGTH(J)
FIRST(J)	RANGE
LAST	SIZE

*** PREDEFINED TYPES**

STRING

RECORD TYPES

* INTRODUCE A COLLECTION
OF (POTENTIALLY) DIFFERENT
COMPONENT TYPES

* SIMPLE RECORD TYPES

type DAY_OF_YEAR is

record

DAY : INTEGER range 1 .. 31;

MONTH : MONTH_NAME;

YEAR : NATURAL;

end record;

type CPU_FLAGS is

record

CARRY : BOOLEAN;

INTERRUPT : BOOLEAN;

NEGATIVE : BOOLEAN;

ZERO : BOOLEAN;

end record;

```
type CPU_STATE is
  record
    PRIORITY : INTEGER range 0 .. 7;
    FLAG      : CPU_FLAGS;
  end record;
```

* DISCRIMINATED RECORDS

```
type SQUARE(SIDE : INTEGER := 4) is
  record
    MATRIX : SIMPLE_ARRAY(1 .. SIDE, 1 .. SIDE);
  end record;
```

```
type TWO_SQUARES(LENGTH : INTEGER) is
  record
    FIRST  : SQUARE(LENGTH);
    SECOND : SQUARE(LENGTH);
  end record;
```


* VARIANT RECORDS

```
type AIRCRAFT_RECORD(KIND : AIRCRAFT_ID := UNKNOWN) is
  record
    AIRSPEED      : SPEED;
    HEADING       : DIRECTION;
    LATITUDE      : COORDINATE;
    LONGITUDE     : COORDINATE;
    case KIND is
      when CIVILIAN =>
        null;
      when MILITARY =>
        CLASSIFICATION : MILITARY_TYPE;
        SOURCE          : COUNTRY;
      when FOE : UNKNOWN =>
        THREAT          : THREAT_LEVEL;
    end case;
  end record;
```

SUMMARY OF RECORD TYPES

* SET OF VALUES

-- COLLECTION OF (POTENTIALLY) DIFFERENT COMPONENTS

* STRUCTURE

-- record

 component_list

end record

WHERE COMPONENT_LIST NAMES THE ELEMENTS OF
THE RECORD

* SET OF OPERATIONS

-- AGGREGATES

-- ASSIGNMENT :=

-- EXPLICIT CONVERSION

-- MEMBERSHIP in not in

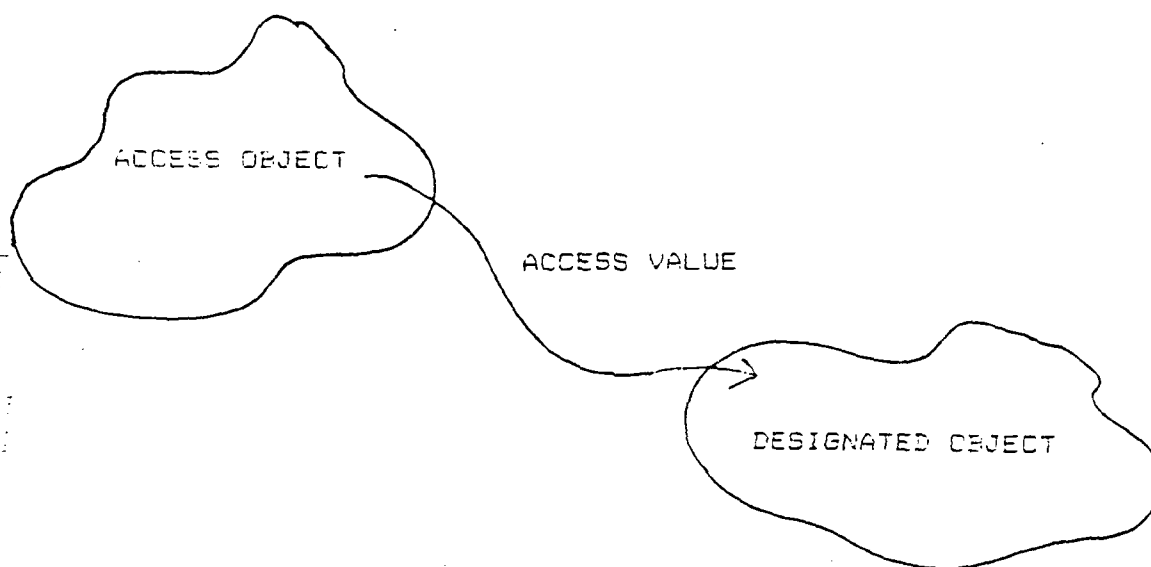
-- QUALIFICATION

-- RELATIONAL = /=

-- SELECTION

* ATTRIBUTES

ADDRESS	LAST_BIT
BASE	POSITION
CONSTRAINED	SIZE
FIRST_BIT	



-- VALUES PROVIDE ACCESS TO OTHER OBJECTS

Figure 8-6: Access Data Type

ACCESS TYPES

- * PROVIDE DYNAMIC ACCESS
TO OTHER OBJECTS

- * SIMPLE TYPES

```
type BUFFER is
  record
    MESSAGE : STRING(1 .. 10);
    PRIORITY : INTEGER range 1 .. 100;
  end record;
type BUFFER_POINTER is access BUFFER;
```

- * INCOMPLETE TYPES

```
type NODE;
type LINK is access NODE;
type NODE is
  record
    LEFT : LINK;
    VALUE : STRING(1 .. 5);
    RIGHT : LINK;
  end record;
```

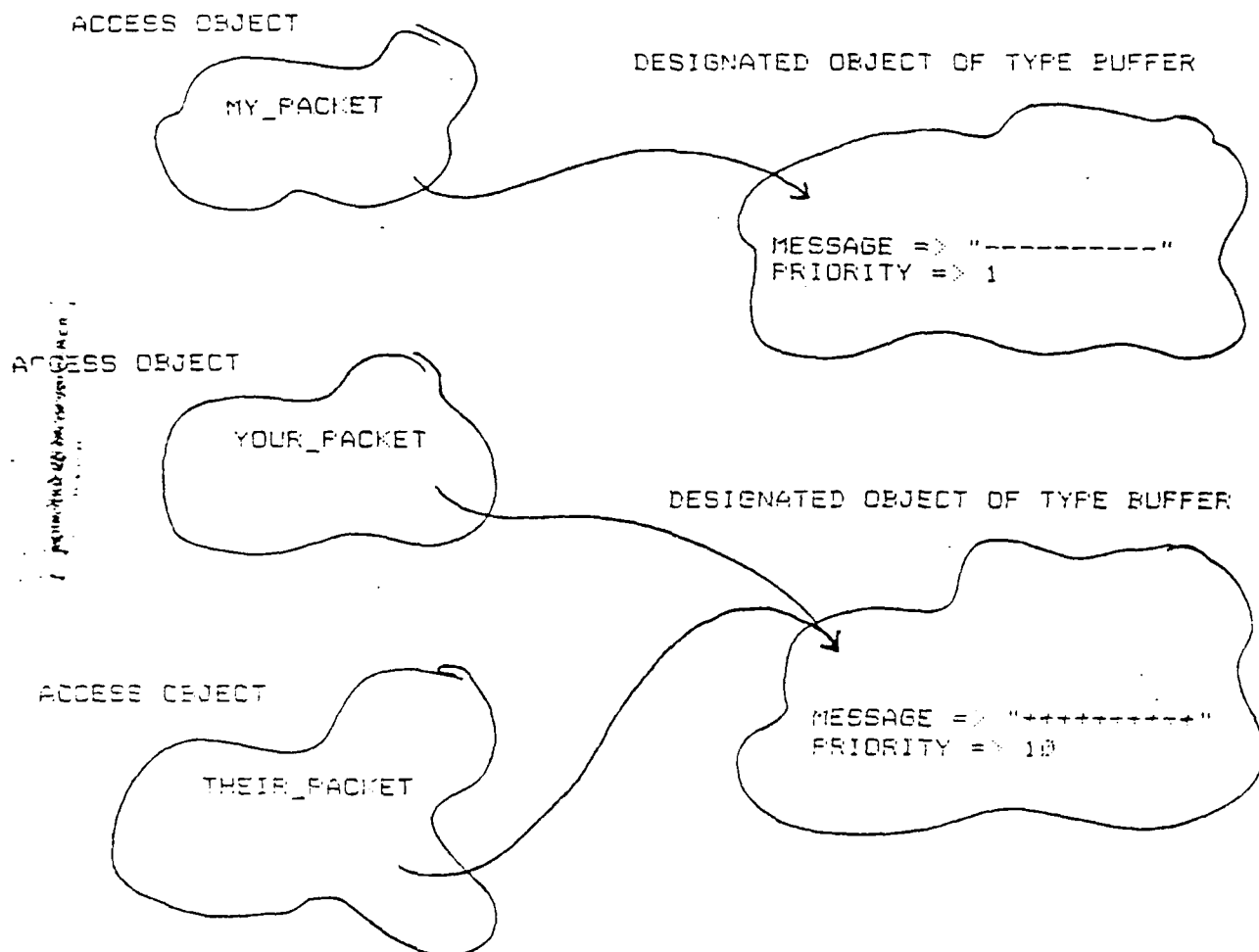


Figure 8-7: Relationship of Access Values and Objects

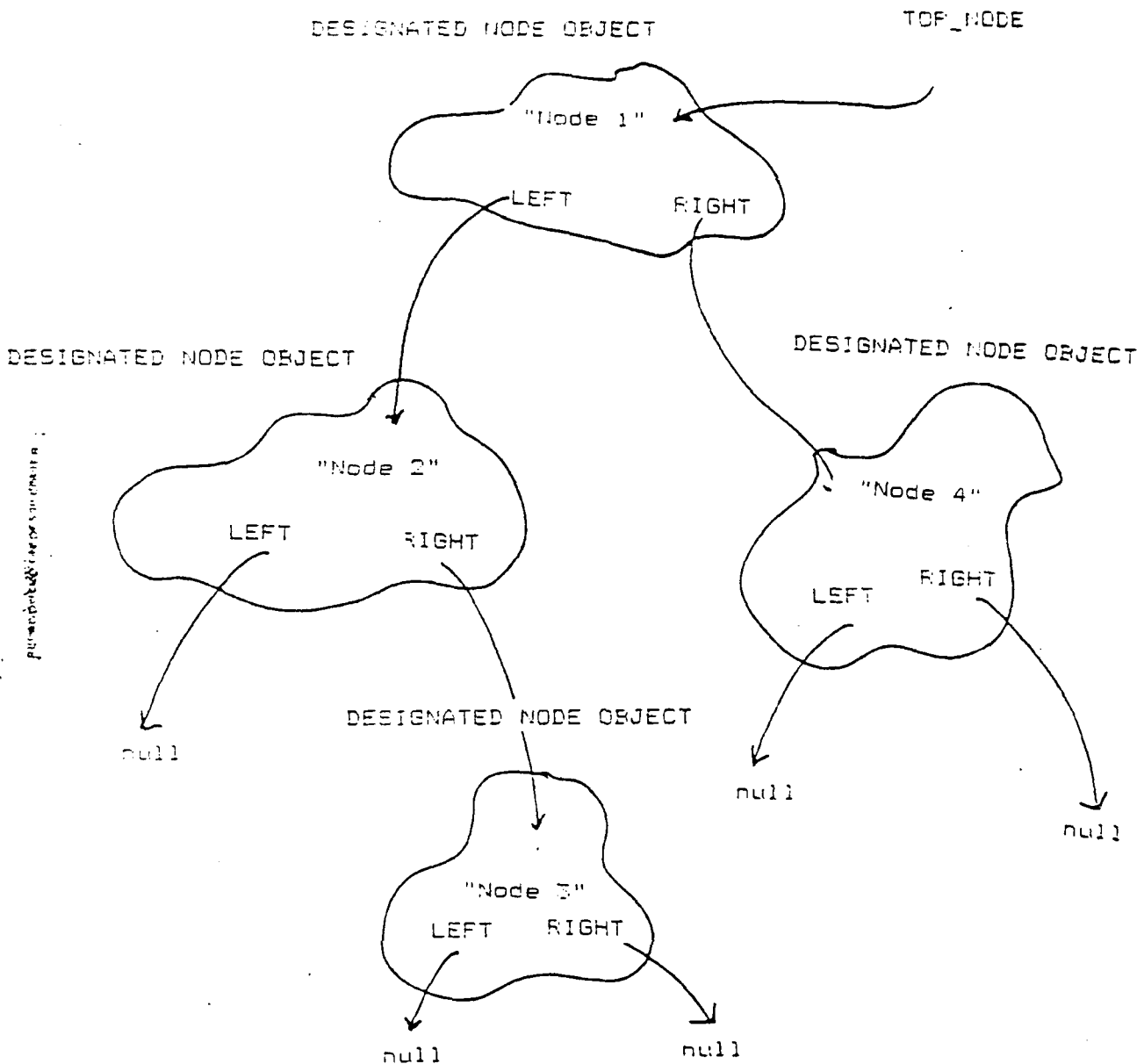


Figure 8-8: Designating the Relationship of Objects

SUMMARY OF ACCESS TYPES

* SET OF VALUES

-- ACCESS VALUES TO DESIGNATED OBJECTS

* STRUCTURE

-- access subtype_indication

WHERE SUBTYPE_INDICATION IS THE TYPE OF
THE DESIGNATED OBJECT

* SET OF OPERATIONS

-- ALLOCATION

-- ASSIGNMENT :=

-- EXPLICIT CONVERSION

-- INDEXING

-- MEMBERSHIP in not in

-- QUALIFICATION

-- RELATIONAL = /=

-- SELECTION

* ATTRIBUTES

ADDRESS SIZE

BASE STORAGE_SIZE

PRIVATE DATA TYPES

* DEFINE ABSTRACT DATA
TYPES WHOSE VALUES ARE
HIDDEN FROM THE USER

* LIMITED PRIVATE TYPES

package PASSWORD is

type VALUE is limited private;

function IS_VALID(CODE : in VALUE)

return BOOLEAN;

procedure SET (CODE : out VALUE;

AUTHORIZATION_LEVEL : in NATURAL);

private

type VALUE is new STRING(1 .. 40);

end PASSWORD;

*** PRIVATE TYPES**

package RANDOM is
 type NUMBER is private;
 procedure SET(SEED : in INTEGER; VALUE : in out NUMBER);
 function UNIFORM_RANDOM return NUMBER;
private
 type NUMBER is
 record
 SEED_VALUE : INTEGER;
 VALUE : FLOAT;
 end record;
end RANDOM;

SUMMARY OF PRIVATE DATA TYPES

* SET OF VALUES

-- HIDDEN FROM THE USER

* STRUCTURE

-- HIDDEN FROM THE USER

* SET OF OPERATIONS

-- EXPLICIT CONVERSION

-- MEMBERSHIP

-- QUALIFICATION

-- FOR LIMITED PRIVATE TYPES

ONLY THOSE OPERATIONS DEFINED IN THE
CORRESPONDING PACKAGE SPECIFICATION ARE
AVAILABLE

-- FOR PRIVATE TYPES

OPERATIONS OF ASSIGNMENT AND TEST FOR
EQUALITY OR INEQUALITY ARE AVAILABLE
IN ADDITION TO THOSE DEFINED IN THE
PACKAGE SPECIFICATION

* ATTRIBUTES

ADDRESS BASE SIZE

SUBTYPES AND DERIVED TYPES

* PROVIDE FURTHER FACTORIZATION OF TYPE CHARACTERISTICS

* SUBTYPES

-- RANGE CONSTRAINT

subtype INDEX is NON_NEGATIVE range 0 .. 10;

-- ACCURACY CONSTRAINT

subtype COARSE is WEIGHT delta 10.0;

-- INDEX CONSTRAINT

subtype VECTOR_3D is VECTOR(1 .. 3);

-- DISCRIMINANT CONSTRAINT

subtype HEAT_SENSOR is SENSOR(KIND => TEMPERATURE);

* DERIVED TYPES

type MASS is new FLOAT;

type WEIGHT is new FLOAT;

type BUDGET is new FLOAT range 0.0 .. 12_000.0;

* TYPE CONSTRAINTS ARE STATIC; SUBTYPE CONSTRAINTS NEED NOT BE STATIC

DECLARATIONS

* SIMPLE DECLARATIONS

DISTANCE : FLOAT;
RESPONSE : CHARACTER;
NUMBER : INTEGER;
GRADES : array(1 .. 100) of FLOAT;

* DECLARATIONS WITH CONSTRAINTS

NAME : STRING(1 .. 40);
BOTTOM : INTEGER range -10 .. -1;

* DECLARATIONS WITH INITIAL VALUES

RANGE : DISTANCE := 0.0;

* CONSTANT DECLARATIONS

FIRST_MONTH : constant MONTH_NAME := JANUARY;
PI : constant := 3.141_592_65;
DIAMETER : constant := 4;

VALUES

* SIMPLE VALUES ARE DENOTED BY LITERALS

-- INTEGER NUMERIC LITERAL 1_024
-- REAL NUMERIC LITERAL 0.398_829_138
-- ENUMERATION LITERAL BLOCKED
-- CHARACTER STRING "WAREHOUSE"
-- NULL ACCESS VALUE null
-- CHARACTER LITERAL 'b'
-- BASED NUMERIC LITERAL 16#1FFE#

* COMPOSITE VALUES ARE DENOTED BY AGGREGATES

EXPRESSIONS

* CREATE NEW VALUES FROM
PRIMARIES AND OPERATORS

* PRIMARIES INCLUDE

-- STRING LITERAL	"prompt"
-- NUMERIC LITERAL	10.125
-- NAME	MATRIX_1
-- ALLOCATOR	new COUNT' (0,0,0,0)
-- FUNCTION CALL	COS(37.5)
-- TYPE CONVERSION	INTEGER(123.9)
-- QUALIFIED EXPRESSION	COEFFICIENT' (0.53)
-- PARENTHESESIZED EXPRESSION	(3 ** 4)
-- NULL VALUE	null

*** BASIC OPERATORS IN
ORDER OF PRECEDENCE ARE**

-- EXPONENTIATING	**		
-- MULTIPLYING	*	/	
	mod	rem	
-- UNARY	+	-	
	not	abs	
-- ADDING	+	-	&
-- RELATIONAL	=	/=	<
	<=	>	>=
-- LOGICAL	and	or	xor
	and then	or else	
-- MEMBERSHIP	in	not in	

SEQUENTIAL STATEMENTS

* ONE STATEMENT IS EXECUTED
AFTER ANOTHER IN A LINEAR
FASHION

* KINDS OF SEQUENTIAL
INCLUDE

-- ASSIGNMENT
-- NULL
-- PROCEDURE CALL
-- RETURN
-- BLOCK

* WE WILL ALSO CONSIDER THE
GOTO IN THIS SECTION

ASSIGNMENT STATEMENTS

- * REPLACE THE CURRENT VALUE
OF A VARIABLE FROM AN
EXPRESSION VALUE
- * TYPE OF BOTH SIDES OF THE
ASSIGNMENT MUST BE
COMPATIBLE
- * ASSIGNMENT STATEMENT EXAMPLES

```
VALVE_RECORD(1 .. 10)      := VALVE_RECORD(6 .. 15);
VOLTAGE_1                   := VOLTAGE_2 + 24.0;
MATRIX_1                   := MATRIX_2;
LOCAL_SCHEDULE.all         := COUNT'(0,0,0,0);
SCHEDULER_TABLE(READY)    := SCHEDULER_TABLE(READY) + 1;
VALVE_RECORD(COUNT_1).OPEN := TRUE;
VALVE_RECORD(1 .. 10)      := VALVES'(1 .. 10 =>
                                (NAME      => "SPARE      ",
                                 POSITION   => "WAREHOUSE ",
                                 OPEN      => FALSE,
                                 FLOW_RATE => 0.0));
```

AD-A142 783

PROCEEDINGS PAPERS OF THE AFSC (AIR FORCE SYSTEMS
COMMAND) AVIONICS STAND..(U) AERONAUTICAL SYSTEMS DIV
WRIGHT-PATTERSON AFB OH DIRECTORATE O..

22

UNCLASSIFIED

C A PORUBCANSKY NOV 82

F/G 9/2

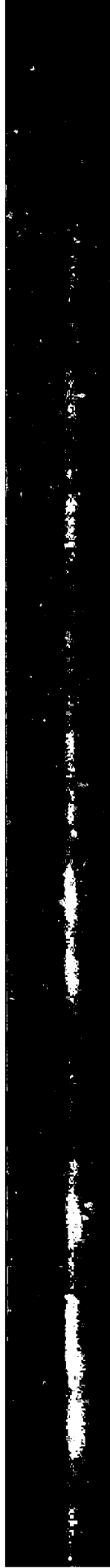
NL

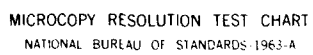
END

DATE _____

FILMED

8-8.





MICROCOPY RESOLUTION TEST CHART
NATIONAL BUREAU OF STANDARDS-1963-A

NULL STATEMENT

* EXPLICITLY STATES INACTION

* NULL EXAMPLE

```
null;
```

RETURN STATEMENT

* RETURN CONTROL FROM A SUBPROGRAM

-- PROCEDURE return;
-- FUNCTION return <expression>;

* A SUBPROGRAM MAY HAVE MULTIPLE RETURNS

* PROCEDURE EXAMPLE

procedure INSTALL (BUFFER : in LINE;
 LIST : in out TABLE;
 TOP : in out POSITIVE) is

begin
 LIST (TOP + 1) := BUFFER;
 for INDEX in 1 .. TOP
 loop
 if LIST (INDEX) = BUFFER then
 return;
 end if;
 end loop;
 TOP := TOP + 1;
end INSTALL;

Expressions and Statements

BLOCK

- * TEXTUALLY ENCAPSULATES A SEQUENCE OF STATEMENTS
- * MAY BE NAMED AND HAVE A LOCAL EXCEPTION HANDLER
- * BLOCK EXAMPLES

```
begin
  A := B/C;
exception
  when NUMERIC_ERROR =>
    A := 0.0;
end;
```

```
SWAP:
  declare
    TEMP : FLOAT;
  begin
    TEMP := VOLTAGE_1;
    VOLTAGE_1 := VOLTAGE_2;
    VOLTAGE_2 := TEMP;
  end SWAP;
```


GOTO STATEMENT

- * UNCONDITIONALLY (AND UNGRACEFULLY) TRANSFER CONTROL

- * GOTO EXAMPLE

```
<<SHUT_DOWN>> START_POWER_DOWN_SEQUENCE;
```

```
...
```

```
goto SHUT_DOWN;
```

CONDITIONAL STATEMENTS

* SELECTION OF ONE OF A
NUMBER OF ALTERNATIVE
SEQUENCE OF STATEMENTS

* CONDITIONAL STATEMENTS
INCLUDE

-- IF

-- CASE

IF STATEMENT

- * SELECTS ONE OR NONE OF
A SEQUENCE OF STATEMENTS
DEPENDING ON THE TRUTH
VALUE OF ONE OF SEVERAL
EXPRESSIONS

- * IF EXAMPLES

```
if COUNT_1 < 0 then  
    COUNT_1 := 0;  
end if;
```

```
if VALVE_RECORD(1).OPEN then  
    VALVE_RECORD(2).OPEN := TRUE;  
    VALVE_RECORD(3).OPEN := FALSE;  
else  
    VALVE_RECORD(2).OPEN := FALSE;  
    VALVE_RECORD(3).OPEN := TRUE;  
end if;
```

Software Engineering with Ada

```
if VOLTAGE_1 > VOLTAGE_2 then
    VOLTAGE_1 := VOLTAGE_2;
elsif VOLTAGE_1 < VOLTAGE_2 then
    VOLTAGE_2 := VOLTAGE_1;
else
    null;
end if;
```

Expressions and Statements

CASE STATEMENT

- * SELECTS ONE SEQUENCE OF STATEMENTS BASED ON THE VALUE OF A DISCRETE EXPRESSION

- * CASE EXAMPLES

```
case PROCESS_STATE is
  when RUNNING => SCHEDULER_TABLE(RUNNING) := 1;
                  IS_ACTIVE := TRUE;
  when READY   => SCHEDULER_TABLE(READY)   :=
                  SCHEDULER_TABLE(READY) + 1;
                  IS_ACTIVE := FALSE;
  when BLOCKED => SCHEDULER_TABLE(BLOCKED) :=
                  SCHEDULER_TABLE(BLOCKED) + 1;
                  IS_ACTIVE := FALSE;
  when DEAD    => SCHEDULER_TABLE(DEAD)    :=
                  SCHEDULER_TABLE(DEAD) + 1;
end case;
```

case COUNT_1 is
 when 1 => VALVE_RECORD(COUNT_1).OPEN := TRUE;
 when 2 | 3 => VALVE_RECORD(COUNT_1).OPEN := FALSE;
 when 5 .. 10 => VALVE_RECORD(COUNT_1).OPEN := FALSE;
 when others => VALVE_RECORD(COUNT_1).OPEN := TRUE;
 VALVE_RECORD(COUNT_1).FLOW_RATE := 1.0;
end case;

ITERATIVE STATEMENTS

- * PERMITS A SEQUENCE OF STATEMENTS TO BE EXECUTED ZERO OR MORE TIMES
- * FORMS OF ITERATION INCLUDE
 - BASIC LOOP
 - FOR LOOP
 - WHILE LOOP
- * ALSO ASSOCIATED WITH THE LOOP STATEMENT IS THE EXIT STATEMENT

LOOP EXAMPLES

```
loop
    GET_SAMPLE;
    PROCESS_SAMPLE;
end loop;
```

```
loop
    GET_SAMPLE;
    exit when TEMP > MAX_TEMP;
    PROCESS_SAMPLE;
end loop;
```

```
OUTER_LOOP:
loop
    ...
    INNER_LOOP:
    loop
        ...
    end INNER_LOOP;
    ...
end OUTER_LOOP;
```


for INDEX in RUNNING .. DEAD

loop

SCHEDULER_TABLE(INDEX) := 0;

end loop;

for INDEX in reverse TOTAL_VALVES

loop

VALVE_RECORD(INDEX).OPEN := FALSE;

end loop;

for INDEX in 1 .. COUNT_1

loop

...

end loop;

for I in VALVES'RANGE

loop

...

end loop;

while (SCHEDULER_TABLE(1).FLOW_RATE > 10.0) and (not IS_EMPTY)

loop

...

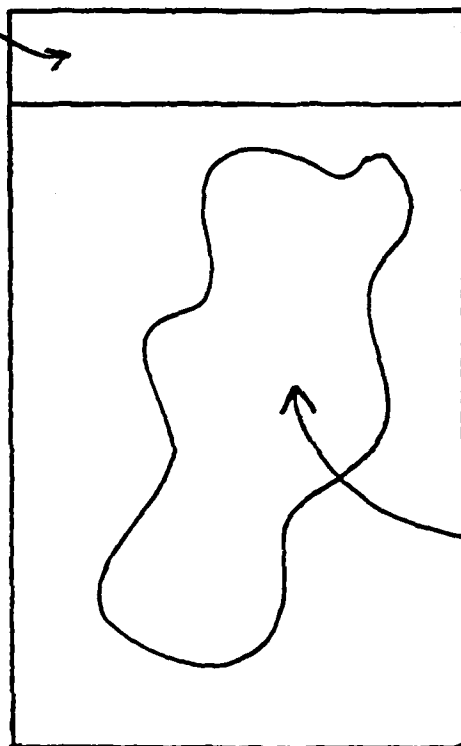
end loop;

SUBPROGRAMS

- * ARE THE BASIC EXECUTABLE UNIT
- * PROVIDE ALGORITHMIC ABSTRACTION
- * ADA SUBPROGRAMS INCLUDE
 - PROCEDURES
 - FUNCTIONS

Software Engineering with Ada

SUBPROGRAM SPECIFICATION



SUBPROGRAM BODY

Figure 10-1: Symbol for an Ada Subprogram

Subprograms

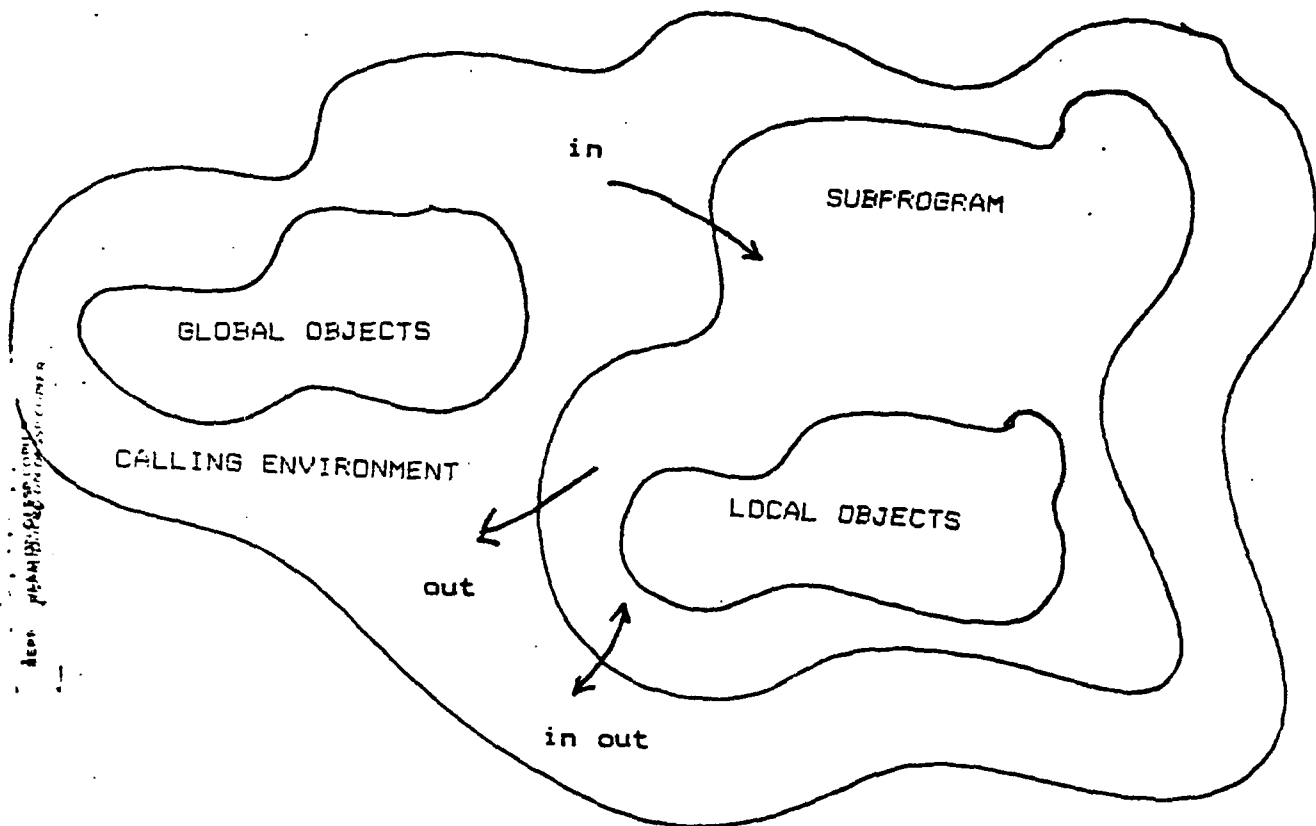


Figure 10-2: Model of an Ada Subprogram

PARAMETER MODES

* IN

- ONLY THE ACTUAL VALUE IS USED; THE SUBPROGRAM CANNOT MODIFY THE VALUE

* OUT

- THE SUBPROGRAM CREATES A VALUE BUT DOES NOT USE THE VALUE OF THE ACTUAL PARAMETER

* IN OUT

- THE SUBPROGRAM USES THE VALUE FROM THE ACTUAL PARAMETER AND MAY ASSIGN A NEW VALUE TO IT

NAMING SPECIFICATIONS

* SIMPLE SPECIFICATIONS

```
procedure COUNT_LEAVES_ON_BINARY_TREE;
procedure PUSH (ELEMENT : in    INTEGER;
               ON      : in out BUFFER);
procedure ROTATE (POINT   : in out COORDINATE;
                 ANGLE    : in    RADIANS);
function COS (ANGLE : RADIANS) return FLOAT;
function "*" (X, Y : in MATRIX) return MATRIX;
function RANDOM return FLOAT;
```

* SPECIFICATIONS WITH DEFAULTS

```
procedure PRINT (BANNER   : in STRING;
                CENTERED  : in BOOLEAN := TRUE;
                SKIP_PAGE  : in BOOLEAN := TRUE);
```

* OVERLOADED SPECIFICATIONS

```
procedure SET (LISTING : in BOOLEAN);
procedure SET (PIXEL   : in COLOR;
               FRAME    : in out BUFFER);
procedure SET (PRIORITY : in NATURAL);
procedure SET (ADDRESS  : in NATURAL);
```

Subprograms

SUBPROGRAM BODIES

- * COMPLETE THE ALGORITHM
INTRODUCED IN THE SPEC-
IFICATION
- * MAY BE SEPARATELY COMPILED
- * TAKE THE FORM

```
specification
begin
    sequence_of_statements
    {exception part}
end;
```

SUBPROGRAM CALLS

* GIVEN THE FOLLOWING PROCEDURES

```
procedure SEARCH_FILE(KEY   : in    NAME;
                      INDEX : out   FILE_INDEX);
procedure SLEEP      (TIME   : in    DURATION := 10.0);
procedure SORT       (DATA   : in out NAMES;
                      ORDER  : in    DIRECTION := ASCENDING);
procedure SORT       (DATA   : in out NUMBERS;
                      ORDER  : in    DIRECTION := ASCENDING);
procedure TURN_ON    (LIGHT  : in    LOCATION);
```

* POSITIONAL PARAMETER CALLS

```
SEARCH_FILE("SMITH, J", RECORD_ENTRY);
SLEEP(120.0);
SORT(PERSONNEL_NAMES, DESCENDING);
SORT(GRADES, ASCENDING);
TURN_ON(OFFICE_LIGHTS);
```

REPRODUCED ON OFFSP COPY R

*** NAMED PARAMETER ASSOCIATION**

```
SEARCH_FILE(KEY => "SMITH, J",  
            INDEX => RECORD_ENTRY);  
  
SLEEP(TIME => 120.0);  
  
SORT(DATA => PERSONNEL_DATA;  
      ORDER => DESCENDING);
```

*** CALLS WITH DEFAULTS**

```
SORT(PERSONNEL_DATA);  
  
SLEEP;
```

*** GIVEN THE FOLLOWING FUNCTIONS**

```
function COS (ANGLE : in RADIANS)      return FLOAT;  
function HEAT(SENSOR : in SENSOR_NAME) return FLOAT;  
function "+" (X, Y : in MATRIX)        return MATRIX;
```

*** SIMILAR OPTIONS APPLY**

```
DISTANCE := LENGTH * COS(30.0);  
  
VALUE    := HEAT(SENSOR => WING_TIP);  
  
SUM      := "+"(FIRST_MATRIX, SECOND_MATRIX);  
  
SUM      := FIRST_MATRIX + SECOND_MATRIX;
```

Subprograms

PACKAGES

- * PERMIT THE COLLECTION OF
GROUPS OF LOGICALLY RELATED
ENTITIES
- * DIRECTLY SUPPORT INFORMATION
HIDING AND ABSTRACTION
- * PERMIT AN INDUSTRY OF
SOFTWARE MODULES

REPRODUCED ON DTS SP-LOM-18

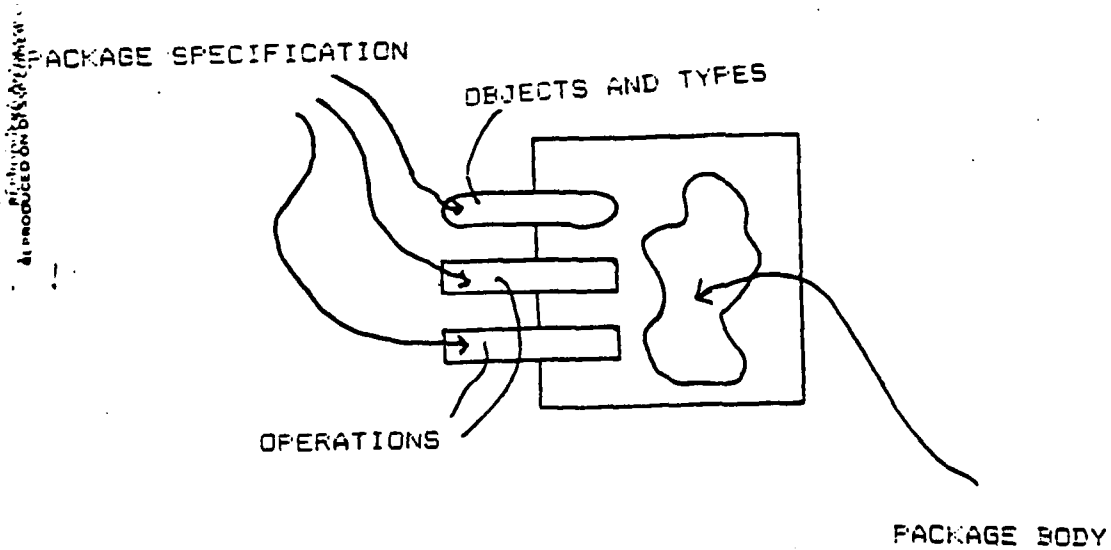


Figure 13-1: Symbol for an Ada Package

PACKAGE SPECIFICATIONS

- * FORM A CONTRACT BETWEEN
THE IMPLEMENTER OF THE
PACKAGE AND THE USER
- * MAY BE SEPARATELY COMPILED

- * TAKE THE FORM

```
package SOME_NAME is
```

```
...
```

```
end SOME_NAME;
```

- * MAY BE FURTHER DIVIDED

```
-- VISIBLE PART
```

```
-- PRIVATE PART
```

PACKAGE VISIBILITY

* GIVEN THE FOLLOWING

package COMPLEX is

type NUMBER is record

REAL_PART : FLOAT;

IMAGINARY_PART : FLOAT;

end record;

function "+"(A, B : in NUMBER) return NUMBER;

function "-"(A, B : in NUMBER) return NUMBER;

function "*" (A, B : in NUMBER) return NUMBER;

end COMPLEX;

REPRODUCED ON DFPSP CORNELL

*** PACKAGES AS DECLARATIVE
ITEMS**

procedure MAIN_PROGRAM is
 procedure FIRST is ... end FIRST;
 package COMPLEX is ... end COMPLEX;
 package body COMPLEX is ... end COMPLEX;
 procedure SECOND is ...
 procedure THIRD is ... end THIRD;
 ...
end SECOND;
begin
 -- sequence of statements
end MAIN_PROGRAM;

*** PACKAGES AS LIBRARY UNITS**

with COMPLEX;
package MAIN_PROGRAM is ...

REPRODUCED ON DTS SP LIP 14

NAMING VISIBLE COMPONENTS

* SIMPLE VISIBILITY

```
with COMPLEX;

procedure SOME_PROGRAM is
    NUMBER_1, NUMBER_2 : COMPLEX.NUMBER;
begin
    ...

    NUMBER_1.IMAGINARY_PART := 37.961;

    NUMBER_1 := COMPLEX."+"(NUMBER_1, NUMBER_2);

    ...
end SOME_PROGRAM;
```

* DIRECT VISIBILITY

```
with COMPLEX;

procedure ANOTHER_PROGRAM is
    use COMPLEX;

    NUMBER_3, NUMBER_4 : NUMBER;
begin
    NUMBER_3 := NUMBER_3 + NUMBER_4;
end ANOTHER_PROGRAM;
```

PACKAGE BODIES

- * COMPLETE THE DECLARATION
OF ENTITIES INTRODUCED
IN THE SPECIFICATION
- * MAY BE SEPARATELY COMPILED
- * TAKE THE FORM

```
package body SOME_NAME is
    ...
end SOME_NAME;
```


PACKAGES AND PRIVATE TYPES

* DEFINE ABSTRACTIONS WHOSE
STRUCTURAL DETAILS ARE
HIDDEN

* TWO CLASSES OF TYPES

-- PRIVATE

-- LIMITED PRIVATE

* PRIVATE TYPE EXAMPLE

package MANAGER is

type PASSWORD is private;

NULL_PASSWORD : constant PASSWORD;

function GET return PASSWORD;

function IS_VALID(P : in PASSWORD) return BOOLEAN;

private

type PASSWORD is range 0 .. 7_000;

NULL_PASSWORD : constant PASSWORD := 0;

end MANAGER;

EXCEPTIONS

- * NAME AN EVENT THAT CAUSES
SUSPENSION OF NORMAL
PROGRAM EXECUTION
- * DRAWING ATTENTION TO THE
EVENT IS CALLED RAISING
THE EXCEPTION
- * THE RESPONSE TO THE EVENT
IS CALLED HANDLING THE
EXCEPTION
- * PERMIT GRACEFUL DEGRADATION

DECLARING AND RAISING EXCEPTIONS

* EXCEPTIONS MAY BE USER- DEFINED

```
ABOVE_LIMITS, BELOW_LIMITS : exception;  
PARITY_ERROR                : exception;  
FATAL_DISK_ERROR            : exception;
```

* SOME EXCEPTIONS ARE PRE- DEFINED

```
-- CONSTRAINT_ERROR  
-- NUMERIC_ERROR  
-- PROGRAM_ERROR  
-- STORAGE_ERROR  
-- TASKING_ERROR
```

* RAISING AN EXCEPTION MAY BE DONE EXPLICITLY

```
raise FATAL_DISK_ERROR;  
raise ABOVE_LIMITS;  
raise;  
raise NUMERIC_ERROR;
```

RAISING PREDEFINED EXCEPTIONS

* CONSTRAINT_ERROR

-- RAISED WHEN A RANGE, INDEX, OR DISCRIMINANT
CONSTRAINT IS VIOLATED

* NUMERIC_ERROR

-- RAISED WHEN A NUMERIC OPERATION YIELDS A
RESULT THAT CANNOT BE REPRESENTED

* PROGRAM_ERROR

-- RAISED WHEN ALL ALTERNATIVES OF A SELECT
STATEMENT HAVING NO ELSE PART ARE ALL CLOSED,
OR IF AN ERRONEOUS CONDITION IS DETECTED

* STORAGE_ERROR

-- RAISED WHEN THE DYNAMIC STORAGE ASSOCIATED
ALLOCATED TO AN ENTITY IS EXCEEDED

* TASKING_ERROR

-- RAISED WHEN EXCEPTION ARISE DURING INTER-
TASK COMMUNICATION

REPRODUCED ON OFFSPR. COPY 12

HANDLING EXCEPTIONS

- * WHEN AN EXCEPTION IS RAISED, PROCESSING IS ABANDONED AND CONTROL PASSES TO AN EXCEPTION HANDLER
- * A HANDLER MAY APPEAR AT THE END OF A BLOCK OR THE BODY OF A SUBPROGRAM, PACKAGE, OR TASK
- * EXCEPTION HANDLERS TAKE THE FORM OF A CASE STATEMENT

ALPRODUCED ON DFS3P COMEN

A TASK

- * IS AN ENTITY THAT OPERATES
IN PARALLEL WITH OTHER
PROGRAM UNITS
- * PHYSICALLY MAY EXECUTE ON
MULTICOMPUTER SYSTEMS,
MULTIPROCESSOR SYSTEMS,
OR WITH INTERLEAVED
EXECUTION ON A SINGLE
PROCESSOR
- * REQUIRES A MEANS FOR INTER-
TASK COMMUNICATION

REPRODUCED ON DSSP FORMER

TASKS

- * PERMIT COMMUNICATING
SEQUENTIAL PROCESSES
- * USE THE CONCEPT OF A
RENDEZVOUS
- * SPECIAL STATEMENTS ARE
PROVIDED FOR TASK CONTROL

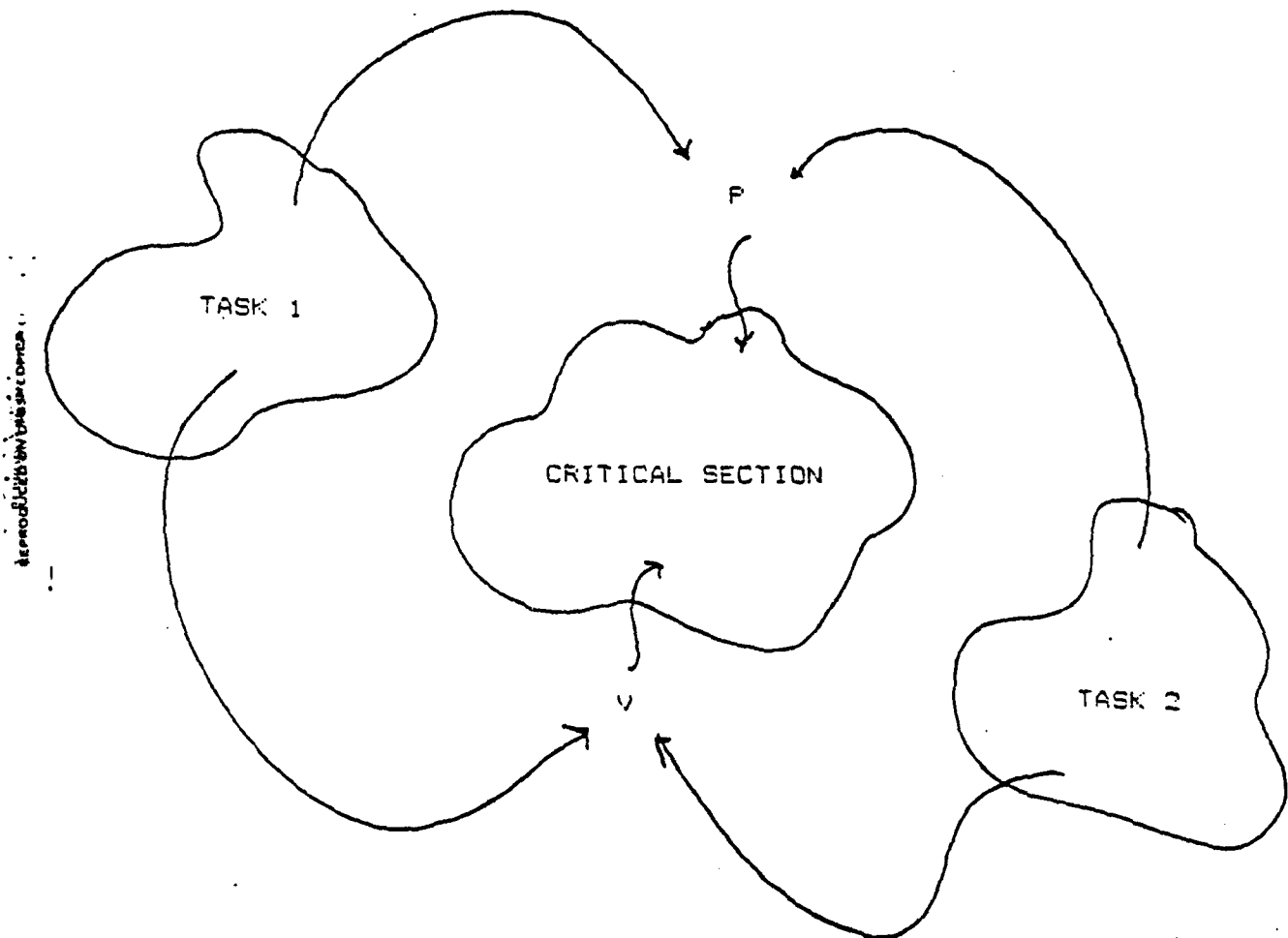


Figure 16-1: Task Communication with Semaphores

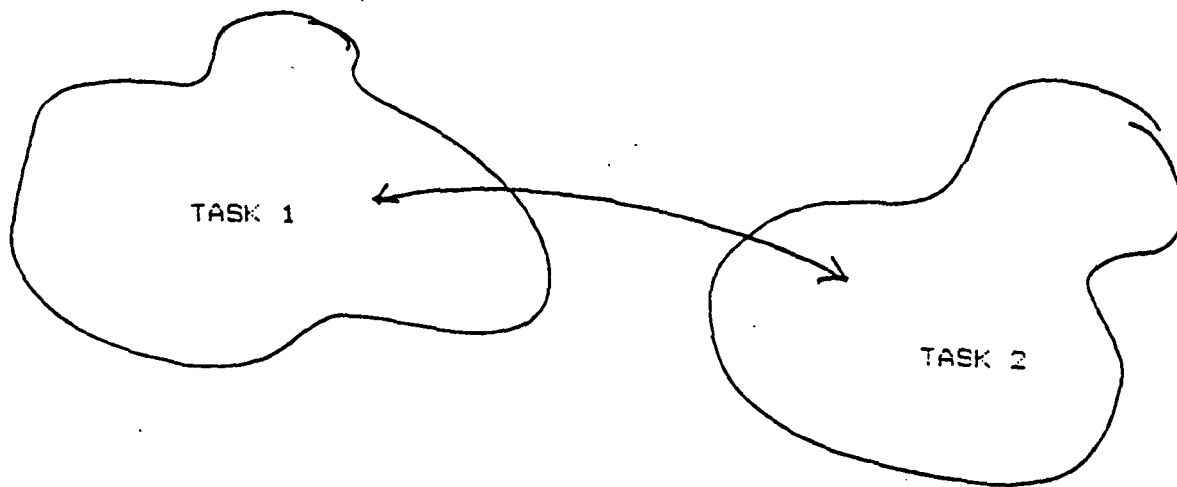


Figure 16-2: Tasks as Communicating Sequential Processes

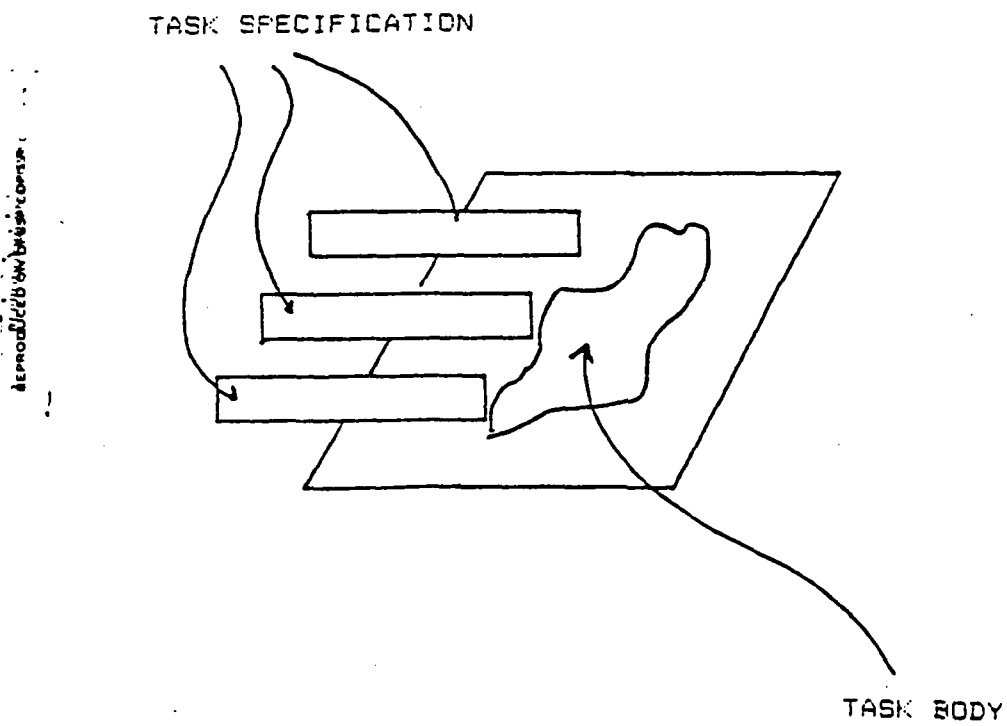


Figure 16-3: Symbol for an Ada Task

THE FORM OF ADA TASKS

- * TASKS ARE ACTIVATED
IMPLICITLY

- * A PARENT TASK WILL NOT
TERMINATE UNTIL ALL OTHER
DEPENDENT TASKS HAVE
TERMINATED
 - A TASK DEPENDS ON AT LEAST ONE MASTER
 - A MASTER IS A TASK, A CURRENTLY EXECUTING BLOCK
OR SUBPROGRAM, OR A LIBRARY UNIT
 - A TASK THAT IS A DESIGNATED ACCESS OBJECT OR
COMPONENT THEREOF, DEPENDS ON THE MASTER THAT
ELABORATED THE ACCESS TYPE
 - ANY OTHER TASK DEPENDS ON THE MASTER WHOSE
EXECUTION CREATED THE TASK OBJECT

TASK SPECIFICATIONS

- * INTRODUCE THE NAME OF THE TASK OBJECT OR TASK TYPE, ALONG WITH VISIBLE ENTRIES

- * MAY NOT BE SEPARATELY COMPILED

- * TAKE THE FORM

```
task SOME_NAME is
    -- TASK ENTRIES
end SOME_NAME;
```

TASK ENTRIES

- * DEFINE THE PATH OF COMMUNICATION WITH A GIVEN TASK
- * HAVE A FORM SIMILAR TO SUBPROGRAM DECLARATIONS
- * SEMANTICS ARE DIFFERENT THAN FOR SUBPROGRAM CALLS

REPRODUCED ON DFPSP CORNER

NAMING VISIBLE COMPONENTS

* GIVEN THE FOLLOWING

```
task PROTECTED_STACK is
  pragma PRIORITY(7);
  entry POP (ELEMENT : out INTEGER);
  entry PUSH(ELEMENT : in  INTEGER);
end PROTECTED_STACK;
```

* NAMING AN ENTRY

```
PROTECTED_STACK.POP(MY_VALUE);
PROTECTED_STACK.PUSH(36);
```

* RENAMING AN ENTRY

```
procedure PROTECTED_POP(ELEMENT : out INTEGER)
  renames PROTECTED_STACK.POP;
```

REPRODUCED ON D5350 COM 14

ENTRY SEMANTICS

- * EACH ENTRY DEFINES AN
IMPLICIT QUEUE

- * ONLY ONE TASK MAY RENDEZVOUS
WITH AN ENTERED TASK AT
A TIME; ALL OTHERS WAIT
IN ORDER OF ARRIVAL IN
THE QUEUE

- * A TASK MAY BE IN ONE OF
FOUR STATES
 - RUNNING (CURRENTLY ASSIGNED TO A PROCESSOR)
 - READY (UNBLOCKED AND WAITING FOR PROCESSING)
 - BLOCKED (DELAYED OR WAITING FOR A RENDEZVOUS)
 - TERMINATED (NEVER OR NO LONGER ACTIVE)

PRIORITY

- * A STATIC VALUE ASSOCIATED WITH EVERY TASK (AND THE MAIN PROGRAM) THAT INDICATES A DEGREE OF URGENCY
- * MAY BE EXPLICITLY SET WITH A PRAGMA
- * DOES NOT AFFECT THE ORDER IN WHICH A QUEUED TASK WILL BE SERVED
- * IF TWO OR MORE TASKS ARE IN THE READY STATE, THE ONE WITH THE HIGHEST PRIORITY WILL BE SELECTED TO RUN

REPRODUCED ON OFFSPR CORP

ASYMMETRY OF TASKS

- * THE CALLER MUST KNOW THE
NAME OF THE SERVER
- * THE SERVER DOES NOT KNOW
THE NAME OF THE CALLER
- * TASKS MAY STILL CALL ONE
ANOTHER MUTUALLY

task FIRST_TASK is
 entry SERVICE;
end FIRST_TASK;

task SECOND_TASK is
 entry SERVICE;
end SECOND_TASK;

task body FIRST_TASK is ...
task body SECOND_TASK is ...

FAMILIES OF ENTRIES

- * DEFINE A SET OF PEER ENTRIES INDEXED BY A DISCRETE VALUE

- * GIVEN THE FOLLOWING

```
type IMPORTANCE is (LOW, MEDIUM, HIGH);  
task MESSAGE is  
    entry GET(IMPORTANCE) (M : out MESSAGE_TYPE);  
    entry PUT(IMPORTANCE) (M : in MESSAGE_TYPE);  
end MESSAGE;
```

- * NAMING A FAMILY MEMBER

```
MESSAGE.GET(HIGH) (YOUR_MESSAGE);  
MESSAGE.PUT(IMPORTANCE => HIGH) (MY_MESSAGE);
```

TASK BODIES

- * DEFINE THE ACTION OF A TASK
- * MAY BE SEPARATELY COMPILED (ONLY AS A SUBUNIT)
- * TAKE THE FORM

```
task body SOME_NAME is
    ...
end SOME_NAME;
```

SAMPLE TASK BODIES

* SIMPLE BODY

```
task WATER_MONITOR;  
  
task body WATER_MONITOR is  
begin  
  loop  
    if WATER_LEVEL > MAXIMUM_LEVEL then  
      SOUND_ALARM;  
    end if;  
  end loop;  
end WATER_MONITOR;
```

REPRODUCED ON Df SSP COPY R

*** BODY WITH AN ACCEPT CLAUSE**

```
task CONSUMER is
  entry TRANSMIT_MESSAGE(M : in STRING);
end CONSUMER;

with LOW_LEVEL_IO;
use LOW_LEVEL_IO;

task body CONSUMER is
begin
  loop
    accept TRANSMIT_MESSAGE(M : in STRING) do
      SEND_CONTROL(MODEM, M);
    end TRANSMIT_MESSAGE;
  end loop;
end CONSUMER;
```

REPRODUCED ON DISK COMPACT

STATEMENTS

* PROVIDE PRIMITIVE FLOW
OF CONTROL

* CLASSES OF STATEMENTS

- SEQUENTIAL
- CONDITIONAL
- ITERATIVE

CONDITIONS FOR RENDEZVOUS

- * AN ENTRY CALL FROM OUTSIDE THE TASK
- * A CORRESPONDING ACCEPT IN THE TASK BODY
- * FOR SIMPLE RENDEZVOUS, A TASK WILL PUT ITSELF TO SLEEP IF IT ARRIVES AT A SYNCHRONIZATION POINT BEFORE ANOTHER
- * WHEN THE RENDEZVOUS IS COMPLETE, THE TWO TASKS ARE RELEASED TO CONTINUE IN PARALLEL

ACCEPT STATEMENTS

- * CORRESPOND TO TASK ENTRIES
- * MUST APPEAR DIRECTLY IN THE TASK BODY
- * OPTIONALLY DEFINE A SET OF STATEMENTS FOR THE RENDEZVOUS ACTION
- * A GIVEN ENTRY MAY HAVE ONE OR MORE CORRESPONDING ACCEPT CLAUSES

*** SAMPLE ACCEPT STATEMENTS**

task SEQUENCER is

entry PHASE_1;

entry PHASE_2;

entry PHASE_3;

end SEQUENCER;

task body SEQUENCER is

begin

accept PHASE_1;

accept PHASE_2;

accept PHASE_3 do

INITIATE_LAUNCH;

end PHASE_3;

end SEQUENCER;

REPRODUCED ON DFPSP COMPILER

TASK STATEMENTS

- * DELAY STATEMENTS

- * STATEMENTS FOR TASK
SYNCHRONIZATION

REPRODUCED ON DASSP COPIER

DELAY STATEMENT

- * SUSPENDS PROCESSING FOR AT LEAST THE GIVEN TIME INTERVAL (IN SECONDS)

- * SIMPLE DELAY STATEMENTS

```
delay 10.0;
```

```
delay DURATION(NEXT_TIME - CALENDAR.CLOCK);
```

- * ADDING NAMED NUMBERS FOR READABILITY

```
SECONDS : constant DURATION := 1.0;
```

```
MINUTES : constant DURATION := 60.0;
```

```
HOURS   : constant DURATION := 3600.0;
```

```
delay 2.0*HOURS + 7.0*MINUTES + 36.0*SECONDS;
```

CLASSIFICATION OF TASKS

* ACTOR TASKS

- HAS NO VISIBLE COMMUNICATION PATHS
- MAY CALL OTHER TASK ENTRIES
- SAMPLE APPLICATION

```
task PRODUCER;
```

* TRANSDUCER TASKS

- HAS VISIBLE ENTRIES
- MAY CALL OTHER TASK ENTRIES
- SAMPLE APPLICATION

```
task MESSAGE_PASSER is
  entry RECEIVE_MESSAGE(M : in MESSAGE);
end MESSAGE_PASSER;
```

* SERVER TASKS

- HAS VISIBLE ENTRIES
- DOES NOT CALL OTHER TASK ENTRIES
- SAMPLE APPLICATION

```
task CONSUMER is
  entry TRANSMIT_MESSAGE(M : in STRING);
end CONSUMER;
```

CLASSES OF TASK COMMUNICATION

* SIMPLE COMMUNICATION

* SELECTIVE RENDEZVOUS BY THE SERVER

- SELECTIVE WAIT
- SELECTIVE WAIT WITH AN ELSE PART
- SELECT WITH GUARDS
- SELECT WITH A DELAY ALTERNATIVE
- SELECT WITH A TERMINATE ALTERNATIVE

* SELECTIVE RENDEZVOUS BY THE CALLER

- TIMED ENTRY CALL
- CONDITIONAL ENTRY CALL

SIMPLE COMMUNICATION

*** AN ACTOR TASK:**

-- CUSTOMER TASK

MAKE_DEPOSIT(ID => 1273,

AMOUNT => 1.0);

* A SERVER TASK

-- TELLER TASK

```
accept MAKE_DEPOSIT(ID      : in INTEGER;
```

AMOUNT : in FLOAT) do

BALANCE(ID) := BALANCE(ID) + AMOUNT;

```
end MAKE_DEPOSIT;
```

REPRODUCED ON DSSP COMPU

SELECTIVE WAIT

* SELECT ONE OF SEVERAL
POSSIBLE ENTRIES

* THE SELECTION IS NON-
DETERMINISTIC

* A SERVER TASK

-- TELLER TASK

loop
 select
 accept MAKE_DEPOSIT(ID : in INTEGER; AMOUNT : in FLOAT) do
 ...
 end MAKE_DEPOSIT;
 or
 accept MAKE_DRIVE_UP_DEPOSIT(ID : in INTEGER; AMOUNT : in FLOAT)
 do ...
 end MAKE_DRIVE_UP_DEPOSIT;
 end select;
end loop;

SELECTIVE WAIT WITH AN ELSE PART

* SELECT ONE OF SEVERAL
POSSIBLE ENTRIES OR AN
ELSE PART IF NO TASKS ARE
WAITING FOR SERVICE

* A SERVER TASK

-- TELLER TASK

loop

select

accept MAKE_DEPOSIT(ID : in INTEGER; AMOUNT : in FLOAT) do

...

end MAKE_DEPOSIT;

or

accept MAKE_DRIVE_UP_DEPOSIT(ID : in INTEGER; AMOUNT : in FLOAT)

do ...

end MAKE_DRIVE_UP_DEPOSIT;

else

DO_FILING;

end select;

end loop;

ALPRODUCED ON DISK CORE II

SELECT WITH GUARDS

- * SELECT ONE OF SEVERAL
POSSIBLE ENTRIES THAT ARE
OPEN BASED ON EVALUATION
OF A GUARD CLAUSE

- * A SERVER TASK

-- TELLER TASK

loop
 select
 when BANKING_HOURS =>
 accept MAKE_DEPOSIT(ID : in INTEGER; AMOUNT : in FLOAT) do
 ...
 end MAKE_DEPOSIT;
 or
 when DRIVE_UP_HOURS =>
 accept MAKE_DRIVE_UP_DEPOSIT(ID : in INTEGER; AMOUNT : in FLOAT)
 do ...
 end MAKE_DRIVE_UP_DEPOSIT;
 else
 DO_FILING;
 end select;
end loop;

SELECT WITH GUARDS

- * SELECT ONE OF SEVERAL POSSIBLE ENTRIES THAT ARE OPEN BASED ON EVALUATION OF A GUARD CLAUSE

- * A SERVER TASK

-- TELLER TASK

loop
 select
 when BANKING_HOURS =>
 accept MAKE_DEPOSIT(ID : in INTEGER; AMOUNT : in FLOAT) do
 ...
 end MAKE_DEPOSIT;
 or
 when DRIVE_UP_HOURS =>
 accept MAKE_DRIVE_UP_DEPOSIT(ID : in INTEGER; AMOUNT : in FLOAT)
 do ...
 end MAKE_DRIVE_UP_DEPOSIT;
 else
 DO_FILING;
 end select;
end loop;

SELECT WITH A DELAY ALTERNATIVE

* SELECT ONE OF SEVERAL
POSSIBLE ENTRIES OR A
DELAY PART IF NO TASKS ARE
WAITING FOR SERVICE

* A SERVER TASK

-- TELLER TASK

loop

select

accept MAKE_DEPOSIT(ID : in INTEGER; AMOUNT : in FLOAT) do

...

end MAKE_DEPOSIT;

or

delay 30*MINUTES;

TAKE_A_BREAK;

end select;

end loop;

REPRODUCED ON DFSSP COPY

SELECT WITH A TERMINATE ALTERNATIVE

* SELECT ONE OF SEVERAL
POSSIBLE ENTRIES OR A
TERMINATE PART

* CONDITION FOR TERMINATION

-- TASK PARENT IS READY TO TERMINATE

-- DEPENDENT TASKS ARE TERMINATED OR READY TO TERMINATE

-- NO CALLING TASKS NEED SERVICE

* A SERVER TASK

-- TELLER TASK

loop

select

accept MAKE_DEPOSIT(ID : in INTEGER; AMOUNT : in FLOAT) do

...

end MAKE_DEPOSIT;

or

terminate;

end select;

end loop;

REPRODUCED ON DFSP COPIER

ALTERNATE FORMS OF TASK TERMINATION

* SIMPLE ABORT

```
abort TELLER;
```

* GIVING A TASK ITS LAST WISHES

```
SHUT_DOWN;  
delay 30*SECONDS;  
abort TELLER;
```

AL PRODUCED ON DFSSA COMPILER

TIMED ENTRY CALL

* ATTEMPT RENDEZVOUS WITH A
SERVER TASK FOR A STATED
MINIMUM TIME

* A CALLING TASK

-- CUSTOMER TASK

select

MAKE_DEPOSIT(ID => 1273, AMOUNT => 1_000.0);

or

delay 10.0*MINUTES;

TAKE_A_HIKE;

end select;

CONDITIONAL ENTRY CALL

* ATTEMPT IMMEDIATE REN-
DEZVOUS WITH A SERVER
TASK

* A CALLING TASK

-- CUSTOMER TASK

select

MAKE_DEPOSIT(ID => 1273, AMOUNT => 1_000.0);

else

RUN_AWAY;

end select;

PROBLEMS WITH CURRENT PROGRAMMING
ENVIRONMENTS

- * LACK OF COMMON INTERFACES
- * INCONSISTANCY AMONG TOOLS
- * LACK OF MEANINGFUL TOOLS
- * INABILITY TO PROCURE PROPER
TOOLS

EXPECTATIONS OF STONEMAN

- * REDUCED COMPILER DEVELOPMENT COSTS
- * REDUCED TOOL DEVELOPMENT COSTS
- * IMPROVED SOFTWARE PORTABILITY
- * IMPROVED PROGRAMMER PORTABILITY

REPRODUCED ON DESP COMET H

ARCHITECTURE OF THE ADA PROGRAMING SUPPORT ENVIRONMENT

* KAPSE

-- KERNAL ADA PROGRAMMING SUPPORT ENVIRONMENT

* MAPSE

-- MINIMAL ADA PROGRAMMING SUPPORT ENVIRONMENT

* APSE

-- ADA PROGRAMMING SUPPORT ENVIRONMENT

Approved on 08/25/2011

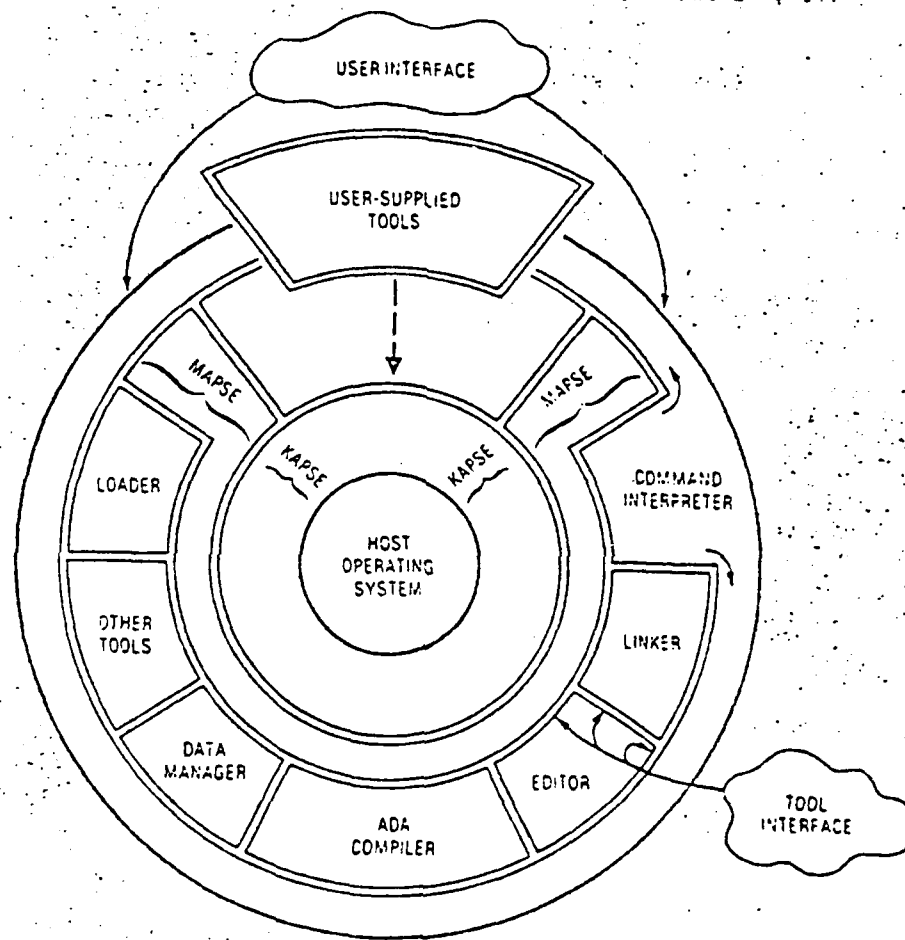


Figure 22-1: The Ada Programming Support Environment (APSE)

THE APSE DATA BASE

- * PROVIDES A REPOSITORY FOR ALL
PROJECT INFORMATION
- * PROVIDES A CENTRAL POINT
OF MANAGEMENT
- * FACILITATES CONFIGURATION
MANAGEMENT

KAPSE

- * IS THE MOST PRIMITIVE LEVEL
OF THE APSE
- * PROVIDES THE LOGICAL TO
PHYSICAL APSE INTERFACE
- * PROVIDES PRIMITIVE ACCESS
TO PROGRAM LIBRARIES

MAPSE

* PROVIDES A ROBUST TOOL SET

- TEXT EDITOR
- PRETTY PRINTER
- COMPILER
- LINKER
- SET-USE STATIC ANALYZER
- CONTROL-FLOW STATIC ANALYZER
- DYNAMIC ANALYSIS TOOLS
- TERMINAL INTERFACE ROUTINES
- FILE ADMINISTRATOR
- COMMAND INTERPRETER
- CONFIGURATION MANAGER

* DIANA CAN PROVIDE A COMMON
INTERFACE AMONG TOOLS

- DESCRIPTIVE INTERMEDIATE ATTRIBUTED
NOTATION FOR ADA

APSE

* PROVIDES TOOLS FOR

- CREATION OF DATA BASE OBJECTS
- MODIFICATION
- ANALYSIS
- TRANSFORMATION
- DISPLAY
- EXECUTION
- MAINTENANCE

* TWO CLASSES OF TOOLS EXIST

- GENERIC TOOLS THAT APPLY TO ALL PROGRAMMING
TASKS WITHOUT REGARD FOR SPECIFIC DISCIPLINES
- METHODOLOGY-SPECIFIC TOOLS THAT SUPPORT A
PARTICULAR PROGRAMMING OR MANAGEMENT DISCIPLINE

Software Engineering with Ada

THE SUCCESS OF ADA

SUCCESS := DESIGN
 + IMPLEMENTATION
 + CLOUT
 + NEED

REPRODUCED ON DTPP CORP

Trends and Conclusion

Software Engineering with Ada

And the Lord said, Behold, the people is one, and they have all one language; and this they begin to do; and now nothing will be restrained from them, which they have imagined to do.

Genesis 11:6

King James Version

REPRODUCED ON D7333 CONT. R

8F

